Skema Penyembunyian Teks Terkompresi *Arithmetic Coding* pada Citra Digital Manggunakan Kuantisasi Berbasis Graf

Elza Oktaviana[1], Adiwijaya[2], Gia Septiana[3]

ejhaokta@gmail.com[1], adiwijaya@ittelkom.ac.id[2], gia.septiana@gmail.com[3]

Fakultas Informatika Universitas Telkom, Bandung

Abstrak

Ilmu dan Seni menyembunyikan data ke media digital merupakan salah satu cara yang biasanya digunakan untuk penyamaran saat melakukan komunikasi yang melibatkan transmisi data. Teknik ini bekerja dengan cara menyisipkan data atau informasi yang bersifat pribadi pada suatu media sehingga data atau informasi yang disisipkan ke media tersebut tidak terlihat secara jelas. Penelitian ini mengajukan sebuah skema penyembunyian data berupa teks pada citra digital dengan menggunakan kuantisasi berbasis graf. Skema ini bekerja dengan cara menyisipkan teks pada suatu graf yang merupakan representasi dari hasil kuantisasi citra digital yang merupakan media penyisipannya. Untuk meningkatkan kapasitas penyisipan, skema ini memanfaatkan algoritma *Arithmetic Coding* untuk kompresi teks yang akan disisipi, dengan tetap memperhatikan kualitas dari citra hasil penyisipan. Hasil penelitian menunjukkan tingkat keberhasilkan skema ini berada pada saat berhasil menyisipkan sekitar 7255 bit data rahasia dengan PSRN citra tersisipi bernilai 28,5324db.

Kata Kunci : *Penyembunyian Data, Kompresi, Kuantisasi berbasis Graf, Vector Quantization, Pewarnaan Graf, Arithmetic Coding,* dan *Genetic Algorithm.*

Abstract

Hiding data is a discipline and art of camouflage by embeding data to a digital media when communicating by involving data transmission. This technique works by embedding personal data or information into a medium, in order the embedded data or information is not visible clearly. This research propose a text hiding scheme into digital image using graph based quantization. This scheme works by embedding the teks on a graph which represents quantization of cover image. This research aims to increasing embedding capacity using Arithmetic Coding to compress the text to be inserted and maintain a good visual quality of image insertion results. From the experimental results, this scheme have a good performances of embedding capasity and PSNR about 7255 bit and 28,5324db.

Keywords *: Data Hiding, Compressing, Graph Based Quantization, Vector Quantization, Graph Coloring, Arithmetic Coding,* and *Genetic Algorithm.*

1. Pendahuluan

Pengguna jaringan komputer dan internet meningkat dibandingkan tahun lalu, dimana Indonesia menduduki peringkat kedua dengan rata – rata peningkatan 18,9% per tahunnya. Dengan perkembangan tersebut, semua yang berhubungan dengan transmisi data digital ataupun multimedia, apalagi yang mengandung data rahasia, membutuhkan keamanan dari berbagai serangan *cyber* [7]. Dalam penelitiaannya, Adiwijaya, dkk [1] mengaplikasikan teknik *watermarking* untuk mempertahankan otoritas kepemilikan dan keaslian gambar medis (orisinalitas) saat menyisipkan data digital (teks atau gambar) ke dalam gambar medis digital asli. Dalam penelitian tentang steganografi, Li, dkk [6] mengaplikasikan metode *Vector Quantization*(VQ) dengan menggunakan *Progressive Exponential Clustering*(PEC) untuk membagi codebook VQ menjadi sekumpulan cluster. Penelitiannya berhasil mengatasi keterbatasan dari kapasitas data yang bisa disisipkan yang merupakan kelemahan dari penelitian In Lin dan Wang sebelumnya. Namun, hasil penelitian Li, dkk, tidak terlalu bagus jika performanya dinilai dari performansi citra stego yang dihasilkan. Dan Yue, dkk [11], berhasil membuat skema yang mampu memberikan kualitas yang baik dari stego-image dan kapasitas penyimpanan yang cukup bagus saat penyisipan data rahasia dengan merepresentasikan codebook VQ ke dalam graf berwarna. Astuti, Widi, yang mengimplementasikan skema Yue dengan menggunakan citra medis mengalami hambatan dalam pembangkitan codebook walaupun kapasitas data yang bisa disembunyikan cukup besar tapi mempengaruhi kualitas citra medis [2]. Sruti, dkk, juga meyarankan metode *Arithmetic Coding* yang mampu memberikan hasil yang sangat bagus untuk mengurangi ruang penyimpanan atau memori yang digunakan untuk penyisipan dan kapasitas transmisi dari informasi [10].

Penelitian ini mengajukan sebuah skema kuantisasi berbasis graf yang merupakan adaptasi dari skema penyembunyian Yue, dkk, Adiwijaya, dan Widi, dkk, dimana *Vector Quantization* (VQ) yang membagun codebook dan agar mendapatkan jumlah warna sebanyak 2$^{\square}$ akan digunakan *Genetic Algorithm* (GA) untuk mewarnai graf yang dihasilkan codebook VQ tersebut. Untuk meningkatkan kapasitas *embedding*, diaplikasikan metode *Arithmetic Coding* dalam kompresi data rahasia berupa text tersebut. Dengan mengimplementasikan skema ini, diharapkan penyembunyian data memberikan performansi yang baik untuk kapasitas penyisipan dan kualitas citra hasil penyisipannya.

2. Landasan Teori

2.1 Penyembunyian Data

Criptography, *watermarking*, dan *steganography* adalah tiga teknik utama yang sering digunakan untuk menyembunyikan data dengan tujuan yang berbeda. *Criptography* adalah cara penyembunyian data dengan mengubah *plain text* (pesan asli) menjadi sebuah *cipher text* (pesan yang sudah dienkripsi)[2]. *Watermarking* adalah cara melindungi data dengan menyisipkan sesuatu ke dalam media yang biasanya dilakukan untuk melindungi hak cipta. Sedangkan konsep *steganography* menonjolkan bagaimana menyisipkan informasi atau data ke media digital (media cover) sehingga data yang disisipkan tersebut tidak terlihat secara jelas. Data yang disembunyikan kadang sama dengan media penyisipannya, tapi banyak penelitian melakukan penyisipan data pada media penyimpan yang berbeda dengan datanya. Contohnya menyembunyikan data rahasia berupa teks pada citra digital, menyembunyikan data pada video, menyembunyikan image pada cover-image, dan lain sebagainya.

2.2 Arithmetic Coding

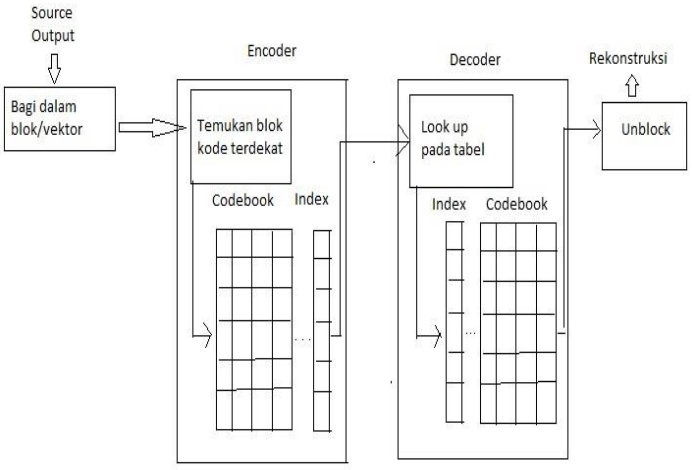
Arithmetic coding adalah metode *lossless compression* (tingkat hilangnya data yang rendah setelah kompresi), metode ini dilakukan dengan cara menghitung frekuensi kemunculan karakter. Karakter yang sering muncul akan disimpan ke dalam bit yang lebih sedikit, dan sebaliknya. Teknik ini dilakukan dengan merombak skema dengan mengganti karakter masukan dengan kode tertentu. Outputnya adalah *floating point* antara 0 sampai 1 yang panjangnya sebanding dengan panjang informasi rahasia yang akan disimpan tersebut. Teknik ini akan sangat berguna ketika kita menemukan kasus "data rahasia" dengan alfabet sedikit, seperti *binary sources* dan alfabet yang memiliki kemungkinan kemunculan yang tinggi.

Berikut tahapan kompresi arithmetic coding:

a. Urutkan nilai probabilitas masing – alfabet.
b. Panggil masing – masing karakter berdasarkan urutan text asli (misalkan: BILL GATES) dan inisialisasikan probabilitas / range nya dalam garis lurus. Karena karakter pertama adalah B (dari BILL GATES), deskripsikan ulang range probabilitas B. Dimana range awal adalah 0,0 – 1,0 menjadi range 0,2 – 0,3 (range alfabet B). Range masing – masing alfabet mengikuti probabilitas yang telah dideskripsikan sebelumnya.
c. Untuk karakter I (karakter ke-dua), yang sekarang memiliki range 0,25 – 0,26 (terhadap range B), lakuakan lagi deskripsi rangenya seperti contoh alfabet B tadi.
d. Proses akan berhenti ketika semua alfabet telah diketahui/ di-replace range barunya. Hasil akhirnya adalah range baru hasil replacement text rahasia tadi.

2.3 Vector Quantization

VQ adalah sebuah teknik kompresi *lossy* yang memanfaatkan fakta bahwa dalam suatu citra, banyak blok yang mirip satu sama lain [11]. Ada tiga langkah utama dalam melakukan *VQ* yaitu pembangkitan *codebook*, *encoding* dan *decoding*. Pada proses pembangkitan *codebook*, berbagai citra dibagi menjadi beberapa vektor latihan dan *codebook* dibangkitkan dari vektor latihan tersebut [3]. Langkah-langkah dalam *encoding* dan decoding seperti yang dijelaskan Sayooo [9] adalah sebagai berikut.



Gambar 2.1 Skema Umum Vector Quantization

Codebook adalah kumpulan *codeword*. Citra dibagi dalam beberapa blok atau vektor yang ukurannya sama dengan *codeword*. *Vector Quantization* memproses setiap blok dengan membandingkan blok tersebut dengan *codeword* yang ada pada *Codebook*. *Codeword* yang paling mirip dengan blok tersebut akan dioutputkan indeksnya pada tabel indeks. *codeword* yang paling mirip adalah *codeword* cw, yang memiliki *Euclidean Distance* terkecil terhadap blok. *Euclidean Distance* dihitung dengan rumus:

$D(\text{blok,cw}) = \sqrt{\sum_{i=0}^{p} (\text{blok}[i] - \text{cw}[i])^2}$ ... .(2.1)

D(blok,cw):*Ecuclidian Distance* blok dan *codeword*
p :Ukuran blok
cw[i] :Nilai pixel ke i pada *codeword*
blok[i] :Nilia pixel ke i pada blok

Setelah memproses setiap blok pada citra, kode kompresi akan terdiri dari dua bagian yaitu *codebook* dan tabel indeks. *Decoder* bisa merekonstruksi citra dengan melakukan *look up* tiap indeks di tabel indeks pada *codebook*.

2.4 Skema Kuantisasi Berbasis Graf

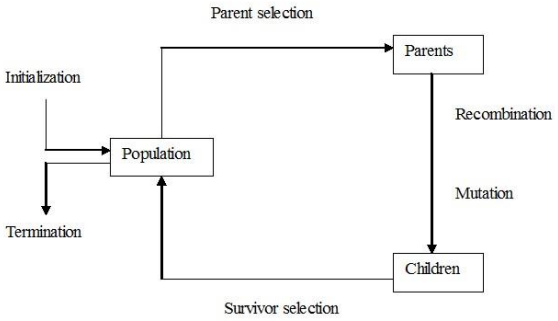
Skema "Kuantisasi Berbasis Graf" merupakan skema penyembunyian data hasil adopsi dari penelitian Yue, dkk yang menggunakan VQ dan merepresentasikan codebook ke dalam graf. Pada penelitiannya, Yue mengkompresi citra cover dengan VQ. Citra cover tersebut dibagi ke dalam beberapa blok, biasanya 4 x 4, atau bahkan ada juga yang membagi ke dalam 32 x 32 blok pixel. Masing – masing blok ditentukan centroidnya yang kemudian disimpan sebagai codeword didalam codebook. Dengan memanfaatkan VQ, codebook akan direpresentasikan ke dalam graf dan diwarnai. Skema ini akan mempertahankan graf bertetangga yang memiliki satu set warna saja (refine graf). Kemudian baru dilakukan penyisipan/ kuantisasi oleh VQ terhadap data rahasia yang telah dikompress ke dalam refine graf. Jadi, kuantisasi graf adalah sebuah skema yang merepresentasikan codebook VQ kedalam sebuah graf untuk nantinya penyisipan dapat dilakukan dengan meng-korespondensikan graf dengan data rahasia yang akan disisipkan.

2.5 Graf

Graph adalah kumpulan verteks dan busur yang menghubungkan antar verteks. Graph coloring adalah teknik mewarnai graph yang terdiri dari beberapa macam: *vertex coloring*, *edge coloring* dan *face coloring*. *Vertex coloring* adalah cara mewarnai graph sehingga setiap ada dua vertex yang saling terhubung oleh suatu edge tidak memiliki warna yang sama. Bilangan kromatik adalah jumlah warna minimal sehingga graph berhasil diwarnai.

2.6 Pewarnaan Genetic Algorithm

Algoritma genetika adalah algoritma komputasi yang diinspirasi teori evolusi yang kemudian diadopsi menjadi algoritma komputasi untuk mencari solusi suatu permasalahan dengan cara yang lebih "alamiah". Salah satu aplikasi algoritma genetika adalah pada permasalahan optimasi kombinasi, yaitu mendapatkan suatu nilai solusi optimal terhadap suatu permasalahan yang mempunyai banyak kemungkinan solusi. Algoritma ini ditemukan di Universitas Michigan, Amerika Serikat oleh John Holland (1975) melalui sebuah penelitian dan dipopulerkan oleh salah satu muridnya, David Goldberg [5]. Secara umum skema Algoritma Genetika sebagai berikut:



Gambar 2.2 Skema umum GA

2.7 Pengukuran Performansi

Pengukuran kemiripan citra tersisipi dengan citra asli secara matematis dilakukan dengan menggunakan *peak signal to-noise ratio*(PSNR) seperti pada penelitian-penelitian lain yang telah dilakukan [1] [2] [3] [11]. Berdasarkan Cheddad [3], PSNR secara matematis dituliskan:

$$\text{PSNR} = 10 * \log_{10}\left(\frac{C_{max}^2}{MSE}\right) \quad \text{.....(2.2)}$$

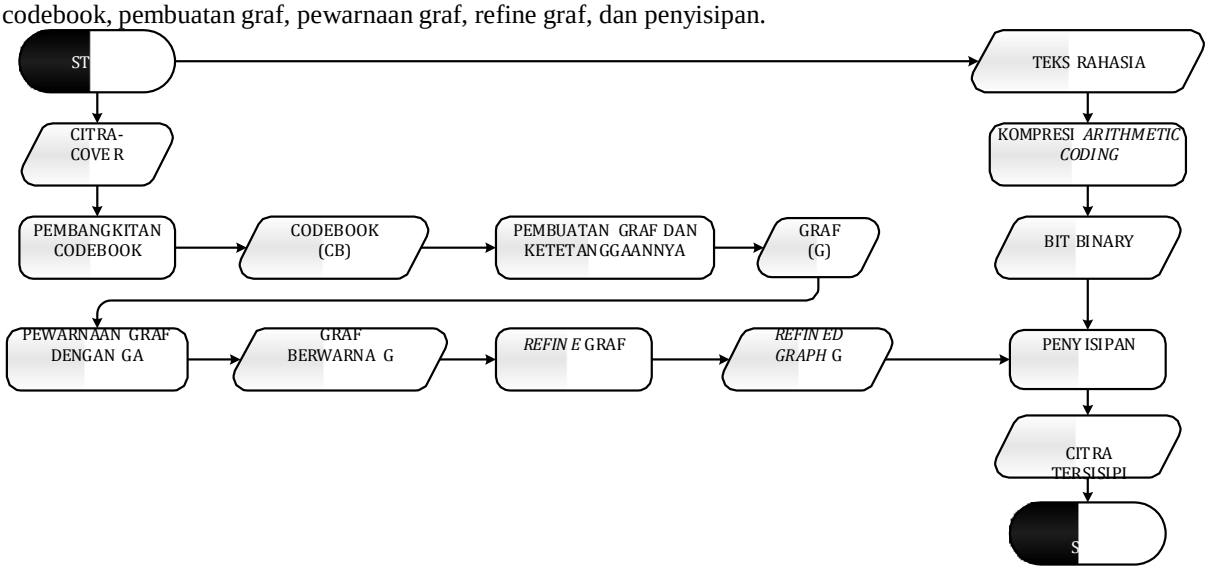
dimana MSE (*Mean square* error) adalah

$$\text{MSE} = \frac{1}{MN}\sum_{x=1}^{M}\sum_{y=1}^{N}(C_{xy} - S_{xy})^2 \quad \text{.....(2.3)}$$

dengan x,y koordinat pixel, M, N adalah ukuran gambar, S citra tersisipi, C citra asli, C_{max} adalah nilai tertinggi dalam gambar asli.

3. Perancangan Sistem

Secara garis besar, proses yang dilakukan pada skema ini ada 6 tahapan, yaitu: kompresi teks, pembangkitan codebook, pembuatan graf, pewarnaan graf, refine graf, dan penyisipan.



Gambar 3.1 Skema umum penyembunyian teks

Tahapan pembangkitan codebook akan menghasilkan citra-cover terkompresi *Vector Quantization* (VQ) atau *VQ-image* dan *codebook* (CB) yang merupakan *centroid* dari hasil klusterisasi vektor set. Awalnya, citra-cover dibagi kedalam blok – blok citra yang berukuran n x n pixel, dimana $n \in \{4, 8, 16, 32, \ldots\}$. VQ akan memanfaatkan algoritma *Linde-Buzo-Gary (LBG)*, yang juga dikenal sebagai *Generalized Lloyd Algorithm (GLA)* untuk membaca blok – blok citra tersebut secara zig – zag sebagai vektor set dari citra-cover yang nantinya akan digunakan sebagai vektor latih x

$$\text{[illegible]} \rightarrow \text{[illegible]} \quad \ldots\ldots (3.1)$$

Jika [illegible] adalah vektor set yang mewakili blok citra ke-[illegible] dimaan ukuran [illegible] adalah [illegible] = [illegible] pixel, dan [illegible] adalah ukuran codebook CB, maka berdasarkan fungsi mapping (3.1) diatas merupakan pemetaan citra-cover untuk dijadikan codebook CB. Sehingga, [illegible] = {[illegible] = 1, 2, 3, ..., [illegible]}, [illegible] = ([illegible], [illegible], ..., [illegible]) ∈ [illegible], [illegible] ∈ {32, 64, 128, ...}, dan [illegible] adalah *Euclidean Distance* antara dua vektor [illegible] dan [illegible]

$$\text{[illegible]} = \sqrt{\sum_{\text{[illegible]}=1}^{\text{[illegible]}} (\text{[illegible]} - \text{[illegible]})^2} \quad \ldots\ldots (3.2)$$

Berikut alur proses pembangkitan *codebook* CB dengan algoritma LBG:

1. Bangkitkan Codebook [illegible], i= 0 secara random dari vektor set [illegible] sebagai *current-codebook*.
2. Untuk masing – masing vektor latih [illegible], lakukan proses klusterisasi seperti dibawah ini:
   a. [illegible] adalah vektor latih [illegible]
   b. Untuk masing – masing [illegible] pada [illegible] hitung *Euclidean Distance* dengan menggunakan fungsi (3.2).
   c. Temukan jarak terdekat vektor [illegible] terhadap [illegible] pada [illegible] misal [illegible] adalah codeword terdekat dari vektor [illegible]
3. Hitung centroid dari kluster dengan menghitung rata – rata pixel dari vektor yang memiliki kluster yang sama. Dan centroid tersebut akan menjadi [illegible] untuk [illegible], dan [illegible] akan menjadi *current cluster*.
4. Jika [illegible] = [illegible] atau iterasi <= [illegible] proses iterasi berhenti. Dan [illegible] akan menjadi codebook [illegible].

Masing – masing codeword [illegible] pada Codebook [illegible] direpresentasikan sebagai vertex [illegible] pada graf [illegible]. Ketetanggaan masing – masing vertex [illegible] pada graf [illegible] ditandai dengan menambahkan *edge* (busur) antara 2 vertex yang memiliki jarak ≤ [illegible], [illegible], [illegible]) ≤ [illegible].

Graf bertetangga [illegible] selanjutnya diwarnai dengan menggunakan *Genetic Algorithm* (GA), dengan pewarnaan 2[illegible]. Pewarnaan menggunakan algoritma ini untuk mendapatkan jumlah warna pada graf sebanyak 2[illegible], dimana n adalah jumlah bit data yang akan disisipkan pada vertex [illegible] di graf [illegible]. Gen ke-i pada kromosom mewakili vertex [illegible] pada graf [illegible], dan best fitness pada proses ini adalah 0. Nilai fitness didapat dari hasil jumlah *bad edge* pada saat pewarnaan graf [illegible]. Bad edge adalah kondisi dimana vertex yang saling bertetangga memiliki warna yang sama.

Tahapan selanjutnya yaitu melakukan *refine* terhadap graf berwarna [illegible]. Proses refine akan menghapus vertex bertetangga yang memiliki bad edge dan vertex bertetangga yang tidak memiliki semua jenis warna. Hasil dari proses ini adalah *Refined Graph* G yang akan digunakan pada proses penyisipan.

Sebelum proses penyisipan dilakukan, pastikan terlebih dahulu teks yang akan disisipkan telah dikonversi ke bilangan bit binary. Bit binary tersebut adalah hasil dari kompresi dengan menggunakan metode *arithmetic coding* (AC). Berikut adalah pseudo code dari algoritma AC ini:

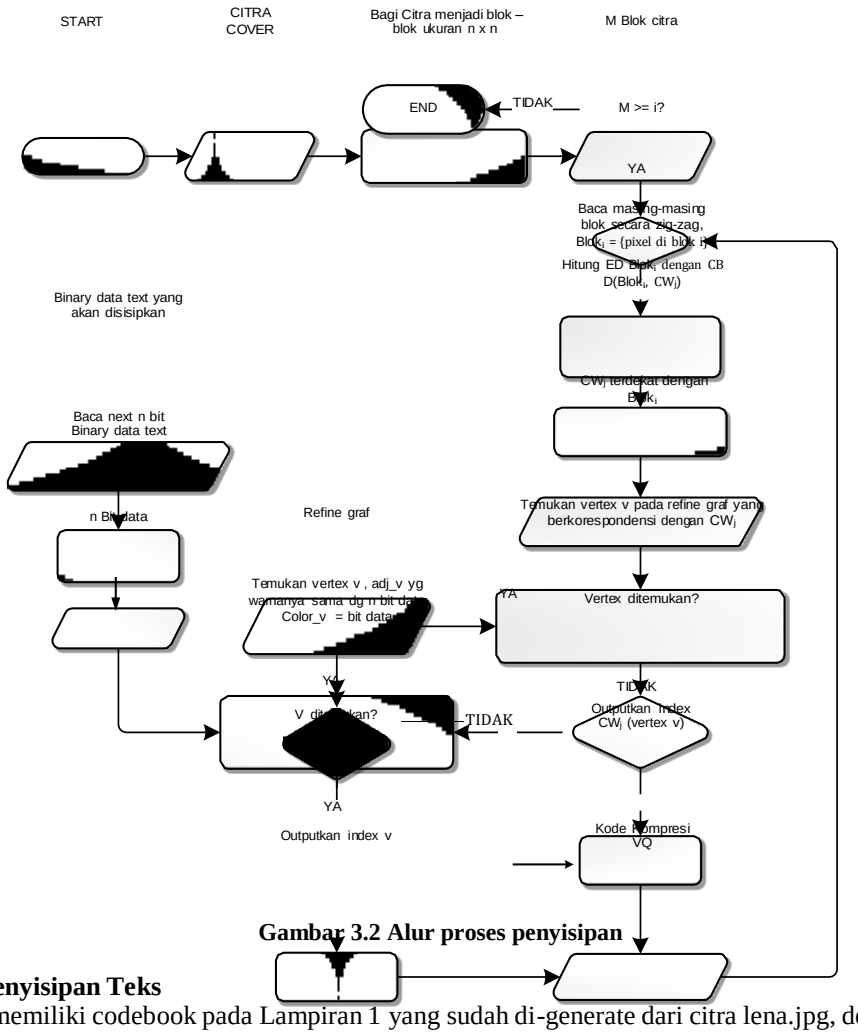
```
[illegible]_(0) ← 0, h_(0) ← 1, [illegible] ← 1
while not EOF (fin), baca karakter ke i
  do [illegible]_(i) ← [illegible]_(i−1) + (h_(i−1) − [illegible]_(i−1)) * [illegible]
     h_(i) ← [illegible]_(i−1) + (h_(i−1) − [illegible]_(i−1)) * h_[illegible]
     [illegible] ← [illegible] + 1
return [illegible]
```

Keterangan:

[illegible]	: File Input
[illegible]	: Batas bawah interval ke-*i*
$h_{(i)}$	: Batas atas interval ke-*i*
[illegible]$_{\text{interval }(i)}$	: batas bawah interval dari karakter ke-*i*
$h_{\text{interval }(i)}$	: batas atas interval dari karakter ke-*i*

Proses penyisipan dilakukan dengan membaca blok – blok pada citra-cover dan menemukan codeword [illegible] yang terdekat dengan blok tersebut. Kemudian dilakukan pencarian vertex yang berkorespondensi dengan [illegible] pada

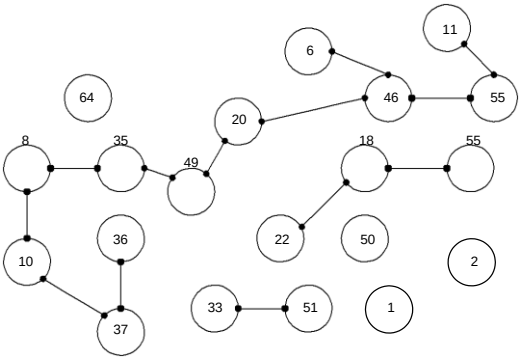
refined graph ��. Jika vertex ditemukan, lakukan pembacaan bit data yang akan disisipi sebanyak n bit dari temukan vertex ��yang memiliki warna yang sama dengan bit data yang akan disisipkan. Pencarian ��dilakukan dengan menggunakan algoritma *Breath First Search* (BFS) pada level atau layar pertama saja. Pencarian pada layar pertama ini dilakukan untuk menjaga jarak penyisipan yang tidak terlalu besar dan menjaga performansi dari citra yang telah disisipi nantinya. Jika ��ditemukan dengan BFS, maka yang menjadi kode kompresi adalah index dari ��tersebut. Dalam kondisi demikian, index dari vertex ��yang terdekat dengan ������akan menjadi kode kompresi dari proses ini. Output dari proses ini adalah kode kompresi.



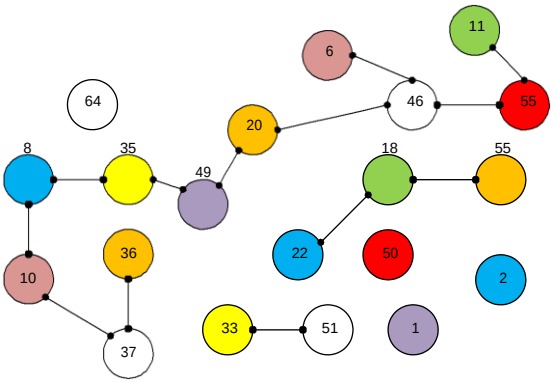
Gambar 3.2 Alur proses penyisipan

4. Contoh Penyisipan Teks

Misalkan kita memiliki codebook pada Lampiran 1 yang sudah di-generate dari citra lena.jpg, dengan *block_size* = 4, *codebook_size* = 64, dan *adj_thresh* = 30, serta bit warna = 3 (8 jumlah warna).

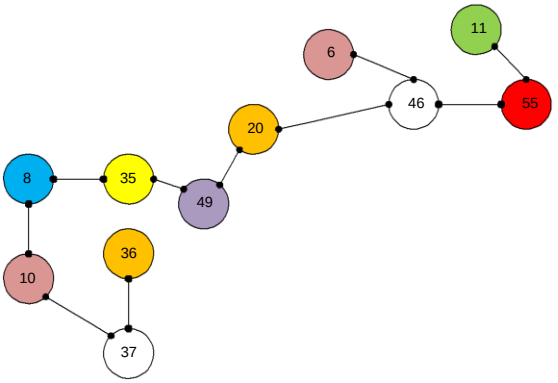


Gambar 4.1 Graf G dari Codebook CB



Gambar 4.2 Graf berwarna G

Vertex pada gambar 4.1 mewakili *codeword* pada *codebook* CB. Busur yang menghubungkan masing – masing vertex merupakan tetangga terdekat yang memiliki jarak < *adj_thresh*. *Genetic Algorithm* berhasil mewarnai graf G dengan pewarnaan 3 bit atau 8 warna seperti Gambar 4.2. Proses *Refine*, akan menghapus vertex bertetangga pada graf G jika ada warna yang tidak ditemukan ditetangganya, dan jika memiliki warna yang sama dengan tetangganya. *Refined graph* G pada Gambar 4.3 adalah contoh dari hasil proses ini.

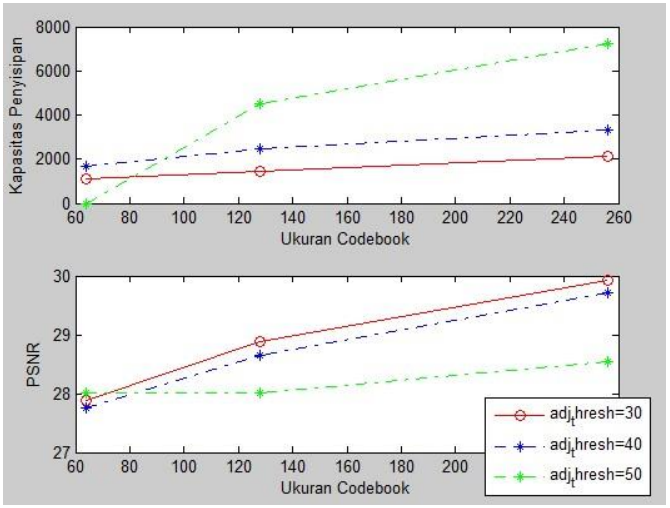


Gambar 4.3 Refined Graph G

Selanjutnya adalah proses penyisipan yang dilakukan dengan membaca *citra-cover* lena.jpg per blok dan mencari jarak terpendek dengan *codeword* pada CB. Misal, $blok_1$ = {179 178 173 176 179 176 173 176 177 175 173 173 174 175 172 171}, *codeword* terdekat dengan ����pada CB min(⍰⍰⍰⍰, ⍰⍰) = 14.3875 yaitu CW_{10} = {178 178 177 177 178 178 178 178 178 179 178 177 178 178 178 177}, misalkan bit data yang dibaca adalah '000' = warna 0. CW_{10} ada di refined graph, dan CW akan mencari vertex tetangganya yang memiliki warna 0. Vertex 37 dan 46 ($ED_{10, 37}$= 29.5973, $ED_{10, 46}$= 92.5311) memiliki warna 0, dengan pencarian BFS, akan ditemukan vertex terdekat dengan CW_{10} adalah CW_{37}. Maka, kode kompresi yang di outputkan adalah 37 (vertex tetangga yang memiliki warna yang merepresentasikan bit yang akan disimpan). Jika $blok_i$ tidak menemukan vertex terdekatnya di refined graph, maka yang menjadi kode kompresi adalah vertex terdekatnya tersebut.

5. Analisis Hasil Pengujian

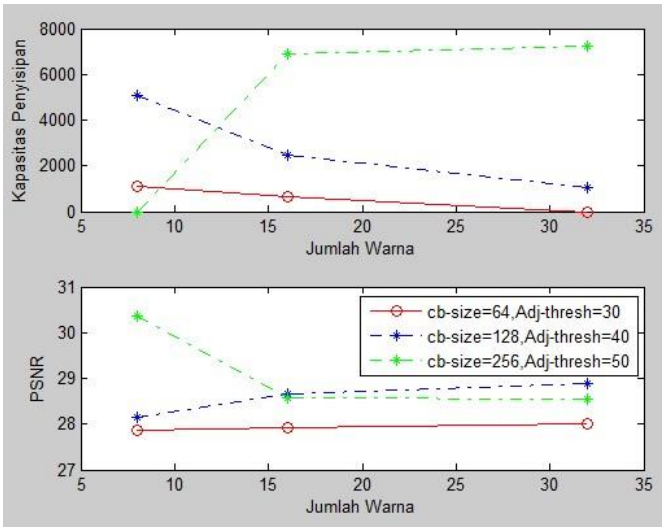
Gambar 7.1 mengilustrasikan variasi kapasitas penyisipan dan PSNR terhadap peningkatan nilai *codebook_size*, *adj_thresh* dan *color_number*, dimana diperlihatkan perubahan kepasitas penyisipan dan PSNR yang relatif meningkat. Peningkatan *codebook_size* sebanding dengan peningkatan jumlah vertex, dan peningkatan nilai parameter *Adj_thresh* akan meningkatkan jumlah vertex bertetangga. Jika vertex – vertex bertetangga ini ada di *refined graf*, maka tidak diragukan lagi kapasitas penyisipan akan semakin besar. Hal tersebut berlaku juga untuk perubahan nilai PSNR yang cenderung naik sebanding dengan peningkatan *codebook_size*-nya.



Gambar 5.1 Pengaruh *codebook_size, adj_thresh, dan color_number*

Berbeda halnya dengan parameter *codebook_size* dan *adj_thresh* yang dapat meningkatkan jumlah vertex pada *refined graph*, nilai *color_number* cenderung akan mengakibatkan perubahan kapasitas penyisipan dan nilai PSNR yang tidak stabil. Contoh pada Gambar 5.1, ketika nilai *codebook_size* = 64, *adj_thresh* = 50, dan *color_number* = 8, kapasitas penyisipan = 0. Selain itu pada Gambar 5.2 skenario pengujian dengan nilai *codebook_size* = 256, *adj_thresh* = 50, dan *color_number* = 8 dan *codebook_size* = 64, *adj_thresh* = 30, dan *color_number* = 32, juga tidak berhasil menyisipkan satu bit pun data, yaitu saat dimana tidak adanya vertex ditemukan di *refined graf*. Kondisi kosongnya vertex di *refined graf* disebabkan oleh 2 hal, yaitu ketika vertex bertetangga memiliki *bad edge* dan ketika vertex bertetangga tidak memiliki jumlah warna yang lengkap.

Pada skenario tertentu, seperti saat banyaknya vertex bertetangga yang akan diwarnai dengan *color_number=8*, ditemukan banyaknya vertex bertetangga memiliki warna yang sama atau best fitness ≠ 0. Dan pada skenario dimana vertex bertetangga sedikit diwarnai dengan 32 warna, refine graf tidak menemukan warna yang lengkap pada vertex bertetangga. Kondisi di atas akan menyebabkan vertex di refined graph kosong, dan tidak ada bit data yang berhasil disisipkan. Nilai PSNR saat kapasitas penyisipan bernilai 0 adalah nilai PSNR dari citra terkompresi VQ.



Gambar 3.3 Pengaruh color_number

Jadi, peningkatan *color_number*, akan meningkatkan kapasitas penyisipan hanya jika diiringi dengan banyaknya jumlah vertex bertetangga di *refined graf*. Namun jika vertex bertetangga ditemukan di refined graph, dengan jumlah vertex yang sama kapasitas penyisipan akan meningkat sebanding dengan peningkatan jumlah warnanya. Misalkan, jika jumlah vertex pada *refined graph* = n, dan jumlah vertex yang tersisipi = m, untuk *color_number* 8 (3 bit), maka jumlah bit yang dapat disisipkan adalah m*3. Dan untuk *color_number* 16 (4 bit), jumlah bit data yang dapat disisipkan akan meningkat, yaitu m*4. Begitu pula dengan nilai PSNR, nilai ini hanya akan

terpengaruh ketika peningkatan *color_number* meningkatkan jumlah vertex pada graf berwarna di *refined graph*, dimana semakin banyak keberagaman warna pada *refined graph*, semakin besar kapasitas penyisipan dan semakin kecil nilai PSNR nya.

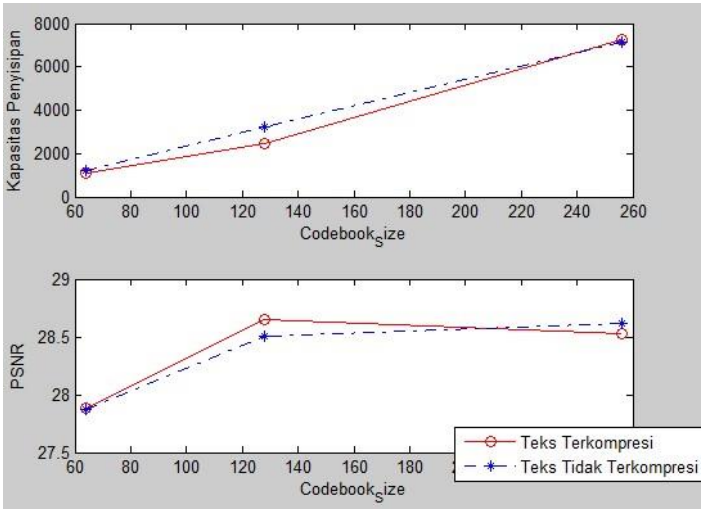
6. Performansi Penyisipan

Dilihat dari waktu proses untuk masing - masing tahapan penyisipan waktu proses terpanjang terjadi pada tahapan pembangkitan codebook, karena pada tahapan ini dilakukan training terhadap vektor set dari blok – blok citra. Namun, pada saat nilai best fitness ≠ 0, waktu pemprosesan pewarnaan akan jauh lebih lama dari pembangkitan codebook. Karena proses hanya akan berhenti jika generasi sudah maksimum. Dan pada tahapan tersebut waktu proses akan pendek jika best_fitness = 0 ditemukan sebelum menyelesaikan n generasi. Namun secara keseluruhan, dengan peningkatan codebook_size, adj_thresh, dan color_number akan meningkatkan waktu pemprosesan yang dibutuhkan untuk skema ini. Berikut gambaran umum perubahan waktu pemprosesan untuk masing – masing tahapan pada skema penyisipan ini:

Tabel 6.1 Hasil Pengujian

CB Size	Adj Tresh	Color	Waktu Pemrosesan (s)			Total Waktu Proses (s)	fitness	PSNR VQ-Code	Kapasitas Penyisipan	PSNR
			Generate Codebook	Generate Graph, Coloring dan Refine	Embed					
Fitness = 0										
64	30	8	143,181	0,813	11,579	155,573	0	28,0084	1116	27,879
128	40	16	284,745	2,047	21,557	308,349	0	29,0274	2456	28,6502
256	50	32	533,71	7,333	39,98	581,023	0	30,3511	7255	28,5324
Fitness > 0										
256	30	8	533,71	426,274	32,453	992,437	17	30,3511	867	30,0234
256	40	8	533,71	680,771	30,633	1245,114	50	30,3511	1311	29,9815
256	50	8	533,71	1157,596	30,633	1721,939	68	30,3511	0	30,3511

Kompresi teks dengan *arithmetic coding* juga dapat mempengaruhi nilai PSNR dan kapasitas teks yang bisa disisipkan. Pengaruh kompresi teks dapat dilihat pada gambar berikut:



Gambar 6.1 Pengaruh kompresi teks

Berdasarkan gambar diatas, dengan atau tanpa kompresi jumlah bit yang berhasil disisipkan tidak mengalami perubahan yang signifikan. Namun, jika dihitung dari jumlah kata atau karakter yang berhasil disisipkan, tentu saja setelah kompresi lebih banyak karakter yang berhasil disisipkan. Jika, jumlah bit dari karakter yang berhasil disisipkan sebanyak n bit, dan dengan ratio kompresi rata – rata *arithmetic coding* 50,59%, maka jumlah karakter yang berhasil disisipkan jika dikompresi akan menjadi 50,59% lebih banyak dari sebelum kompresi.

Walaupun jumlah karakter yang berhasil disisipkan setelah kompresi lebih banyak, nilai PSNR nya lebih kecil daripada sebelum kompresi. Hal tersebut dapat dilihat pada gambar 4.5 yang memperlihatkan perbedaan PSNR sebelum dan setelah kompresi terhadap teks walau tidak terlalu signifikan.

Nilai PSNR adalah nilai yang didapat dengan membandingkan citra asli dengan citra yang terkompresi VQ (cover image) atau dengan citra yang telah tersisipi bit teks. Dengan kata lain, PSNR adalah indikator perubahan kualitas citra. Semakin tinggi nilai PSNR, semakin kecil distrorsi atau perubahan citra setelah tersisipi dengan citra asli, dan semakin menyerupai citra asli. Berdasarkan hasil pengujian parameter – parameter diatas, dapat diketahui bahwa nilai PSNR berbanding terbalik dengan kapasitas penyisipan. Artinya, semakin banyak bit yang berhasil disisipkan, semakin kecil nilai PSNR yang dihasilkan, dan sebaliknya. Berikut pada ditampilkan citra hasil pengujian terhadap parameter - parameternya.

Citra Asli

Citra Terkompresi VQ
Codebook_size: 256
PSNR: 30,3511 db

Citra Tersisipi Data
Codebook_size: 128
Adj_tresh: 50
Color_number: 8
Kapasitas:7788 bit
PSNR: 27,3198 db

Citra Tersisipi Data
Codebook_size: 256
Adj_tresh: 50
Color_number: 32
Kapasitas: 7255 bit
PSNR: 28,5324 db

Citra Tersisipi Data
Codebook_size: 128
Adj_tresh: 50
Color_number: 16
Kapasitas: 4488 bit
PSNR: 27,9837 db

Citra Tersisipi Data
Codebook_size: 128
Adj_tresh: 30
Color_number: 16
Kapasitas: 4180 bit
PSNR: 29,6782 db

7. Kesimpulan

Berdasarkan analisa terhadap pengaruh parameter – parameter pengujian yang dijelaskan sebelumnya, ketiga parameter tesebut tidak dapat mempengaruhi kapasitas penyisipan secara langsung. Karena ketiga parameter tersebut saling terkait dan sangat dipengaruhi oleh proses *refine*. Disamping itu, keserangaman codebook, ketanggaan graf, dan PSNR ditentukan oleh seberapa bagusnya klusterisasi terbentuk dan seberapa tepat centroid yang didapatkan. Penambahan *codebook_size*, penambahan *adj_thresh*, atau peningkatan *color_number* yang menyebabkan jumlah vertex yang bertetangga meningkat memang bisa meningkatkan kapasitas penyisipan, namun bisa saja kapasitas penyisipan menjadi 0 atau menurun hanya karena graf tidak memiliki nilai fitness=0 atau tidak ditemukan di *refined graf.* Dengan kata lain, peningkatan kapasitas penyisipan sangat besar kemungkinannya meningkat, hanya jika mendapatkan kondisi yang tepat dalam ketetanggaan dan pewarnaannya. Dan kembali lagi hal tersebut dipengaruhi oleh codebook yang berhasil dibangkitkan. Jadi, semakin bagus codebook yang dihasilkan, semakin besar kemungkinan ketetanggaannya dan semakin bagus graf berwarna yang terbentuk, sehingga akan semakin banyak vertex yang ditemukan di *refined graph*. Berikut beberapa hal yang dapat disimpulkan dari hasil pengujian skema ini:

a. Kapasitas yang dihasilkan dari pengujian relatif bagus dengan nilai parameter – parameter tertentu.
b. Besarnya kapasitas penyisipan tidak menjamin nilai PSNR yang bagus untuk skema dengan parameter masukannya karena nilai PSNR berbanding terbalik dengan kapasitas penyisipan.
c. Pemilihan skenario terbaik ditentukan berdasarkan orientasi hasil yang diinginkan. Pencapaian penelitian ini adalah *codebook_size=256, adj_thresh=50, color_number=32*, dengan nilai PSNR: 28,5324 db dan Kapasitas bit yang bisa disisipkan adalah 7255 bit, dimana citra dapat disisipkan dengan kualitas yang dapat diterima atau nilai PSNR yang tidak terlalu kecil.

8. Saran

Seperti yang dijelaskas pada analisis sebelumnya, kapasitas terbaik yang dihasilkan pada skema ini tidak memberikan nilai PSNR yang cukup bagus kapasitas bit yang disisipkan tidak cukup besar. Oleh karena itu penulis mengharapkan pengembangan berikutnya sebagai berikut:

a. Mendapatkan hasil dimana akan didapatkan kapasitas maksimum dengan nilai PSNR yang cukup bagus.
b. Dibutuhkan sebuah Metode Cluster yang tepat untuk menghasilkan centroid dari codebook.
c. Diberikan batas jarak pencarian v' (vertex yang bersesuaian dengan bit yang akan disisipkan), atau nilai bfs_thresh untuk tetap menjaga kualitas citra hasil penyisipan.

Referensi

[1] Adiwijaya, W.A.B Wirayuda, S.D. Wiinanjuar, dan U. Muslimah. (2012). "The Multiple Watermarking on Digital Medical Image for Mobility and Authenticity". Springerlink: Operations Research Proceedings 2012, pp 457-462.

[2] Astuti, Widi, Adiwijaya, dan Untari Novia Wisesty. 2015. "Data Hiding Scheme on Medical Image Using Graph Coloring".*The Third International Conference on Science & Engineering in Mathematics, Chemistry, and Physics*, pp 263-268.

[3] Cheddad, Abbas, dkk. (2010). "Digital Image Steganography: Survey and analysis of current methods". Signal Processing, 90 (3), 727-752.

[4] Chuen Lu, Tzu, dan Ching-Yun Chang. (2010). "A Survey of VQ Codebook Generation". Ubiquitous International. Journal of Information Hidding and Multimedia Signal Processing 2010, 1 (3).

[5] Hindi, Musa M., dan Roman Y. Yampolskiy. (2012). "Genetic Algorithm Applied to the Graph Coloring Problem". J.B. Speed School of Engineering: Kentucky.

[6] Li, Yue, dan Chang-Tsun Li. (2006). "Steganographic Scheme for VQ Compressed Images Using Progressive Exponential Clustering". Proceedings of the IEEE International Conference on Video and SignalBasedSurveillance(AVSS'06), 85-90.

[7] Lu, Z.M., dan S.H. Sun. (2000). "Digital image watermarking technique based on Vector Quantization". ELECTRONlCS LETTERS , 36 (4).

[8] Moore, E. Dafid, dan Farid Ahmed. (2003). "Reliable Transmission of Security-enabled Multimedia over Internet". Proc of SPIE, 84 – 91.

[9] Sayooo, Khalid. (2006). "*Introduction to Data Compression"* (3rd Edition ed.). San Fransisco, California: Elseveir Inc.

[10] Sruti, S., Gomathymeenakshi M., B. Karthikeyan, Meka Nayana. (t.thn). "An Efficient Arithmetic Coding Data Compression with Steganography". School of Computing, SASTRA University.

[11] Yue, Shuai, Zhi-Hui Wang, dan Chin-Chen Chang. 2013. "An Image Data Hiding Scheme Based on Vector Quantization and Graph Coloring." *Recent Advances in Information Hiding and Applications* (Springer): 1-17.

Lampiran 1

Index Codebook CB

1	{	163	171	178	185	163	170	177	184	164	170	175	184	163	169	176	184 }
2	{	170	142	109	95	175	151	115	98	173	158	119	100	169	159	128	103 }
...																	
10	{	178	178	177	177	178	178	178	178	178	179	178	177	178	178	178	177 }
...																	
37	{	184	186	187	186	184	185	186	186	183	185	185	186	183	184	185	186 }
...																	
46	{	155	155	155	155	155	155	155	155	155	155	155	154	154	154	154	154 }
...																	
64	{	122	134	147	153	122	136	146	153	123	136	145	152	125	137	144	151 }