SKEMA PENYEMBUNYIAN TEKS TERKOMPRESI *ADAPTIVE HUFFMAN* PADA CITRA DIGITAL MENGGUNAKAN KUANTISASI BERBASIS GRAF

Melanida Tagari[1], Adiwijaya[2], Gia Septiana[3]

Prodi S1 Teknik Informatika, Fakultas Teknik, Universitas Telkom
[1]melanida.tagari@gmail.com, [2]kang.adiwijaya@gmail.com, [3]gia.septiana@gmail.com

Abstrak

Jaringan komputer dan internet semakin banyak digunakan untuk aktivitas pengiriman data. Namun, tidak ada jaminan bahwa jaringan komputer dan internet yang digunakan sebagai media pengiriman data ini aman dari pihak ketiga yang tidak memiliki hak akses terhadap data tersebut [1]. Berbagai teknik telah dikembangkan untuk melindungi data dari pengaksesan secara ilegal. Salah satu diantaranya yaitu dengan menyisipkan/ menyembunyikan data tersebut ke dalam media *cover*. Pada penelitian ini, implementasi penyembunyian data memanfaatkan kuantisasi berbasis graf, yaitu menggunakan *Vector Quantization* (VQ) dan pewarnaan graf dengan menggunakan *Genetic Algorithm*. Untuk meningkatkan kapasitas penyisipan, data dikompres terlebih dahulu dengan menggunakan *Adaptive Huffman* sebelum penyisipan dilakukan. Hasil pengujian menunjukkan bahwa skema ini dapat menghasilkan kapasitas penyisipan sebanyak 9000 bit atau sekitar 1800 karakter, dengan nilai PSNR 27,5054 db.

Keywords: Penyembunyian Data, Kuantisasi Berbasis Graf, Vector Quantization, Pewarnaan Graf, Adaptive Huffman, Genetic Algorithm

Abstract

Computer network and internet have been rapidly used for data transmissions. However, there is no guarantee that the network and the internet used for the transmission media are safe from unauthorized party [1]. Many techniques have been developed to protect the data from illegal access. One of these techniques are by hiding or embedding the data to a cover media. In this research, the data hiding is implemented using graph based quantization which utilizes Vector Quantization and graph coloring using Genetic Algorithm. To increase the hiding capacity, the scheme will compress the data using Adaptive Huffman before it is embedded to the cover media. The testing result shows that the scheme can embed 9000 bits data which is about 1800 characters with PSNR score is 27,5054 db.

Keywords: Data Hiding, Graph Based Quantization, Vector Quantization, Graph Coloring, Adaptive Huffman, Genetic Algorithm

1. Pendahuluan

Pengiriman data digital semakin banyak dilakukan melalui internet, baik data biasa maupun data rahasia. Permasalahan yang muncul kemudian adalah tidak adanya jaminan keamanan pada pengiriman data melalui media ini [1]. Oleh karenanya, berbagai cara dikembangkan untuk melindungi data dari pengaksesan secara ilegal dan satu diantaranya adalah dengan menyembunyikan data tersebut pada sebuah media penyisipan.

Tiga aspek penting pada penyembunyian data adalah: kapasitas, keamanan, dan kekuatan. Kapasitas mengacu pada seberapa besar jumlah data yang dapat disisipkan, keamanan mengacu pada seberapa aman kerahasiaan data dapat terjaga sehingga data tidak dapat diakses oleh *unauthorized party*, dan kekuatan (*robustness*) mengacu pada ketahanan penyembunyian/penyisipan data dari upaya modifikasi terhadap data tersebut (data tidak dapat diubah atau dirusak). Steganografi dan *watermark* adalah teknik yang digunakan untuk menyembunyikan data pada sebuah media *cover*. Bila tujuan utama dari *watermarking* adalah *robustness* dengan tanpa mengurangi kualitas media penyisipan, maka tujuan utama dari steganografi adalah keamanan dan kapasitas [9].

Teknik penyembunyian data, baik pada steganografi maupun *watermarking*, banyak memanfaatkan *Vector Quantization* (VQ) yang termasuk kategori kompresi *lossy*. VQ memanfaatkan fakta bahwa banyak bagian pada citra yang mirip atau redundan. Tiap bagian tersebut kemudian dikuantisasi berdasarkan kemiripannya. Yue, dkk [15] mengajukan skema steganografi berbasis *Vector Quantization* dan pewarnaan graf. Penggunaan VQ dan pewarnaan graf juga telah dimanfaatkan pada penelitian sebelumnya untuk penyembunyian data ke dalam citra medis digital, dengan pewarnaan graf menggunakan PSO [2]. Pada hasil penelitian tersebut, citra yang telah disisipi data memiliki perbedaan yang cukup terlihat dibandingkan dengan citra aslinya. Skema penyembunyian data yang digunakan pada penelitian ini mengadopsi skema yang diajukan Yue, dkk tersebut dengan pewarnaan graf menggunakan *Genetic Algorithm*. Adapun untuk pemrosesan data sisipan, digunakan kompresi *Adaptive Huffman*. David Solomon menjelaskan bahwa pada algoritma *Huffman* biasa, kompresi dilakukan berdasarkan

frekuensi kemunculan simbol pada teks yang akan dikompresi, sedangkan pada prakteknya, informasi ini jarang tersedia. *Adaptive Huffman* memberikan alternatif solusi bagi permasalahan tersebut dengan penggunaan *tree* yang dinamis (di-*update* setiap kali pembacaan simbol dilakukan) [12]. Ida Mengyi Pu juga menyatakan dalam bukunya, *Fundamental Data Compression*, bahwa *Adaptive Huffman* menghasilkan hasil kompresi yang baik [10]. Dengan penggunaan metode ini, skema penyembunyian data diharapkan dapat menghasilkan performansi yang baik berdasarkan kapasitas data yang dapat disisipkan dan kualitas citra hasil penyisipan.

2. Penyembunyian Data

Teknik penyembuyian data secara umum terdiri dari steganografi dan *watermarking*. Kata steganografi (*steganography*) diambil dari bahasa Yunani, yaitu *steganos* yang berarti tersembunyi atau rahasia, dan *graphy* yang berarti tulisan atau gambar [13]. Tujuan utama dari steganografi adalah menyembunyikan data pada sebuah media agar keberadaan data tersebut tidak dapat terdeteksi secara langsung, sedangkan *watermarking* merupakan teknik penyembunyian data dengan media penyisipan sebagai objek yang dilindungi. *Watermarking* biasa digunakan untuk perlindungan hak cipta, dan identifikasi keaslian dokumen digital [5].

3. Kompresi *Adaptive Huffman*

Secara sederhana, kompresi data adalah pemampatan data. Terdapat dua proses utama yang dilakukan pada kompresi data, yaitu *encoding* dan *decoding*. *Encoding* adalah pemrosesan data menjadi representasi data terkompres dengan harapan data hasil kompresi memiliki ukuran yang lebih kecil dibandingkan data asli, sedangkan *decoding* adalah proses rekonstruksi representasi data terkompres ke dalam bentuk yang sama atau mirip dengan data asli sebelum dikompres.

Huffman adalah metode kompresi yang cukup efisien dari segi penggunaan *memory*. Pada kompresi dengan menggunakan metode *Huffman* biasa, baik *encoder* atau pun *decoder* membutuhkan pengetahuan mengenai frekuensi kemunculan dari setiap simbol pada data yang akan diproses [12]. Jika informasi ini tidak ada, maka ada dua proses yang harus dilalui untuk melakukan *encoding*: 1) mengumpulkan data statistik simbol, 2) melakukan *encoding*.

Pada *Adaptive Huffman*, di awal proses transmisi, baik *encoder* maupun *decoder* tidak mengetahui informasi statistik dari simbol-simbol pada data yang akan diproses. *Tree* pada *encoder* maupun *decoder* berisi sebuah *node* yang berkorespondensi dengan seluruh simbol yang belum ditransmisikan (NYT: *Not-Yet-Transmitted*). *Node* tersebut diberi bobot 0. Pada saat proses transmisi berlangsung, setiap *node* yang berkorespondensi dengan simbol yang ditransmisikan akan ditambahkan ke dalam *tree* dan *tree* akan dikonfigurasi dengan prosedur *update*.

Sebelum transmisi dilakukan, *encoder* dan *decoder* menentukan *fix code* untuk setiap simbol. Misalnya, sebuah data teks berisi *unique symbol*: (a_1, a_2, …, a_n), kemudian *fix code* untuk setiap *unique symbol* ditentukan dengan rumusan berikut:

$$n = \sum_{i=0}^{\ell} 2^i, \qquad 0 \le r < 2^\ell$$

$$k = \begin{cases} r - 1 & \text{untuk } 1 \le r \le 2r \\ r - 1 & \text{untuk } 2r < r \le n \end{cases} \quad \text{..... (2.1)}$$

Keterangan:
n = jumlah *unique symbol*;
k = nomor simbol

Contoh:
Jika terdapat 26 *unique symbol* pada data yang akan dikompres, maka:
$n = 26$
$r = \log_2 n = \log_2 26) = 4$
$r = n - 2^r = 26 - 2^4 = 10$

Dengan demikian, *fix code* untuk masing-masing *unique symbol* adalah:

a_1: 00000	a_9: 00010	a_{11}: 1010	a_{16}: 1111
a_2: 00001	….	….	

Pada awal pemrosesan data, *tree* berisi sebuah *node* yang disebut *node* NYT (*not-yet-transmitted*) yang diberi bobot 0 dan nomor *node* ($2n - 1$). *Node* ini berkorespondensi dengan simbol-simbol yang belum dibaca. Ketika sebuah dimbol dibaca untuk yang pertama kalinya, maka akan diberi kode untuk NYT yang diikuti *fix code* untuk simbol tersebut, kemudian sebuah *external node* yang berkorespondensi dengan simbol tersebut

ditambahkan pada *tree*. Jika simbol tersebut sudah pernah dibaca sebelumnya, maka kode kompresi untuk simbol tersebut adalah jalur yang dilalui dari *root* ke *node* yang berkorespondensi dengan simbol tersebut.

Tree akan di-*update* setiap kali sebuah simbol dibaca. Prosedur *update* digunakan untuk menjaga properti *siblings* (*node* turunan dari *parent* yang sama) pada *tree*, yaitu aturan penomoran dan pembobotan *node* yang terurut dari kiri ke kanan, dari bawah ke atas.

4. Kuantisasi Berbasis Graf

Kuantisasi berbasis graf yang dimaksudkan pada penelitian ini adalah penggunaan *Vector Quantization* dan pewarnaan graf pada tahap penyisipan teks ke dalam citra digital. *Vector Quantization* pada citra digital menghasilkan *codebook* berisi *codewords.* Dimana setiap *codeword* akan direpresentasikan sebagai *vertex* pada graf yang kemudian diwarnai dengan menggunakan *Genetic Algorithm*.

4.1 *Vector Quantization* (VQ)

Proses yang dilakukan pada *Vector Quantization* diawali dengan mengkonstruksi sejumlah *codebook* yang masing-masing terdiri dari sejumlah *codeword*, dimana setiap *codeword* merepresentasikan satu blok piksel citra [8]. Kemudian dicari *codeword* pada *codebook* yang paling mirip dengan blok yang sedang diproses. Hal ini ditentukan dengan mengukur *Euclidian distance* yang diformulasikan sebagai berikut:

$$D(\text{blok},\text{cw}) = \sqrt{\sum_{i=0}^{p} (\text{blok}[i] - \text{cw}[i])^2} \quad \text{.....(2.2)}$$

Keterangan :
D (blok,cw) : nilai *Euclidian distance*
blok : blok yang diinputkan/yang akan diproses
cw : *codeword* di dalam *codebook*
blok[i] : nilai *pixel* ke i pada blok input
cw[i] : nilai *pixel* ke i pada *codeword*
p : ukuran blok

Setelah kesemua piksel blok pada selesai diproses, kode kompresi akan memiliki dua bagian: *codebook* dan tabel indeks. Proses *decoding* nantinya akan menggunakan *codebook* dan tabel indeks ini. Gambar di bawah ini mengilustrasikan contoh proses kompresi dengan VQ ini.

4.2 *Genetic Algorithm* (GA)

Genetic Algorithm (GA) mengadopsi proses genetik pada makhluk hidup, yaitu perkembangan generasi dalam sebuah populasi yang terjadi secara alami mengikuti proses seleksi alam, yang terbaik akan bertahan. *Genetic Algorithm* banyak digunakan untuk menyelesaikan permasalahan terkait optimasi, seperti penjadwalan mata kuliah, penyusunan rute kunjungan wisata, *multy traveling salesman problem*, dan sebagainya.

Pada *Genetic Algorithm*, solusi dari suatu masalah dipetakan menjadi kromosom. Kromosom berisi sejumlah gen yang menggambarkan variabel-variabel keputusan yang digunakan dalam solusi. Setiap kromosom memiliki nilai *fitness*. Nilai ini digunakan untuk menilai seberapa bagus sebuah kromosom untuk dimasukan ke dalam GA. Nilai *fitness* ini merepresentasikan kelayakan sebuah alternatif solusi untuk dipilih. Kemudian dengan menirukan proses genetik, GA akan menghasilkan kromosom terbaik (solusi yang mendekati optimum) setelah melewati sekian generasi.

5. Pengukuran Performansi

Parameter yang akan digunakan pada pengukuran performansi untuk skema penyembunyian ini adalah kapasitas dan kualitas citra hasil penyisipan (kemiripan dengan citra asli). Kapasitas penyisipan diukur berdasarkan jumlah bit data yang dapat disisipkan, sedangkan kualitas citra hasil penyisipan diukur berdasarkan kemiripannya dengan citra asli dengan menggunakan *peak signal to-noise ratio* (PSNR).

$$\text{PSNR} = 10 \log_{10} \frac{I_{max}^2}{MSE}, \text{ dengan} \quad \text{.....(2.3)}$$

$$MSE = \frac{1}{MN} \sum_{x=1}^{M} \sum_{y=1}^{N} (I_{xy} - S_{xy})^2 \quad \text{.....(2.4)}$$

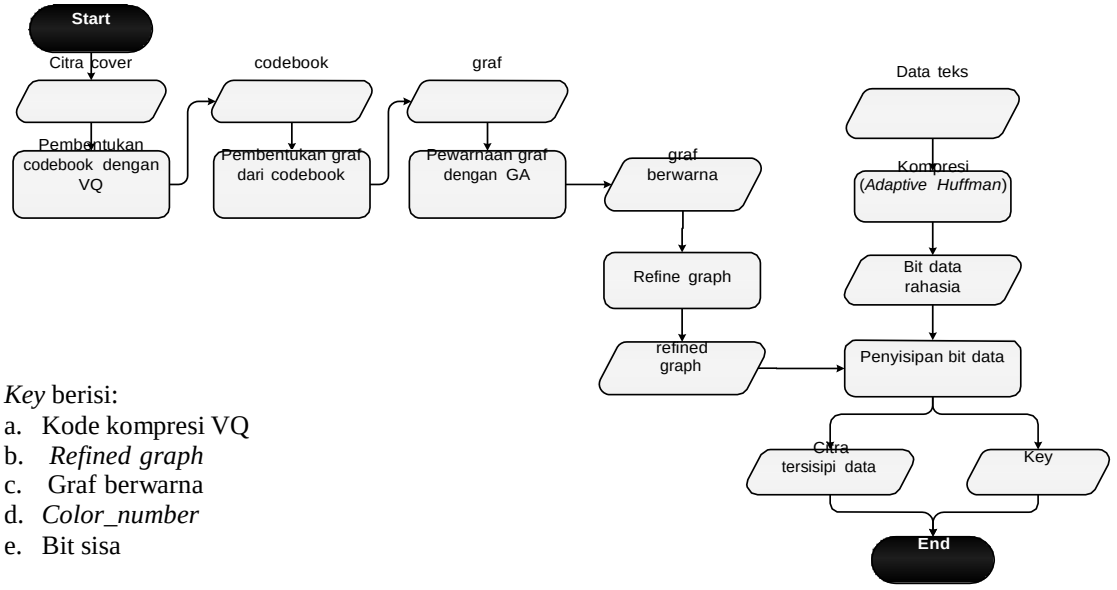
Keterangan:

MSE : *Mean Square Error*
x,y : koordinat
M, N : ukuran citra
S : citra hasil penyisipan

C : citra asli
C_{max} : nilai tertinggi pada citra asli

6. Skema yang Diajukan

Pada tahap penyisipan, *input* berupa data teks dan citra *cover*. Sebelum dilakukan penyisipan ke dalam citra, teks dikompresi terlebih dahulu dengan menggunakan *Adaptive Huffman*. kemudian proses penyisipan data dilakukan melalui beberapa tahap, yaitu pembentukan *codebook* dengan menggunakan VQ, pembentukan graf berdasarkan *codebook* tadi, pewarnaan graf dengan GA, *refine graph*, dan penyisipan bit data terkompresi ke dalam citra . Alur proses pada tahap penyisipan ditunjukkan pada Gambar di bawah ini.

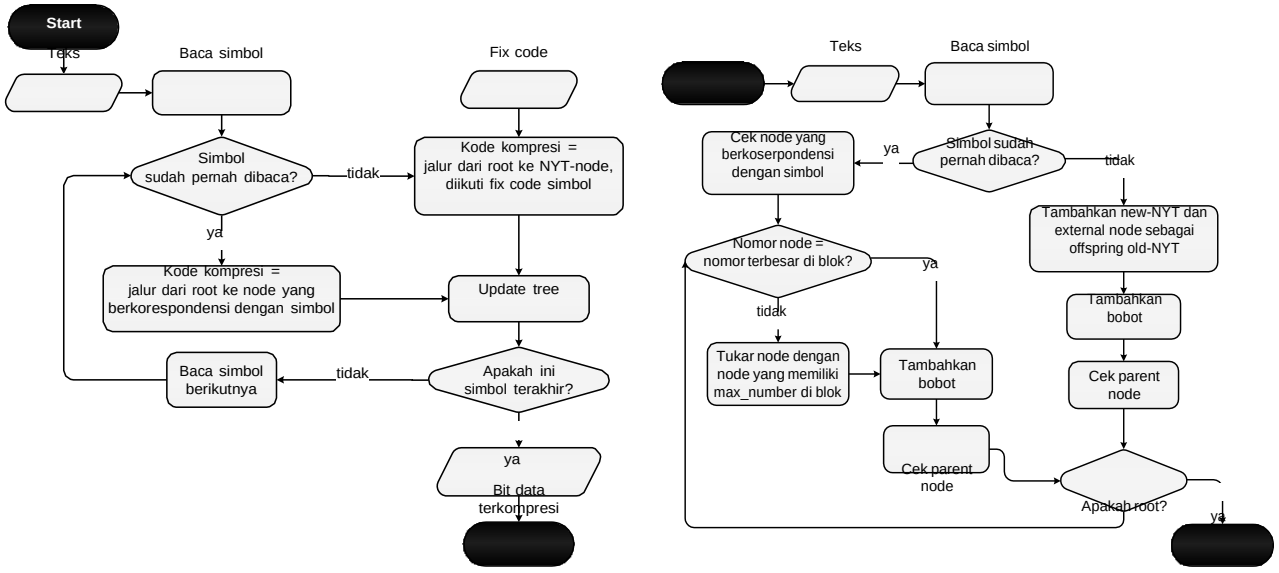


Key berisi:
a. Kode kompresi VQ
b. *Refined graph*
c. Graf berwarna
d. *Color_number*
e. Bit sisa

Gambar 1. Alur Proses pada Tahap Penyisipan

6.1 Kompresi Teks dengan Adaptive Huffman

Kompresi dilakukan untuk meningkatkan kapasitas data yang dapat disisipkan ke dalam media *cover*. Pada penelitian ini, teknik kompresi yang digunakan adalah *Adaptive Huffman*. Pada *Huffman* biasa, kompresi dilakukan berdasarkan probabilitas kemunculan simbol pada teks yang dikompresi. *Tree* akan dibangun dengan memanfaatkan informasi statistik ini dan *tree* bersifat statis. Sedangkan pada *Adaptive Huffman*, *tree* dibuat lebih dinamis. *Tree* terus di-*update* setiap kali pembacaan simbol dilakukan. Pada *Adaptive Huffman*, dua parameter ditambahkan pada *tree*, yaitu: bobot dan nomor. Bobot pada *external node* mendefinisikan jumlah pembacaan suatu simbol, sedangkan bobot pada *internal node* adalah jumlah bobot anaknya (*offspring*) [6]. Berikut adalah skema proses kompresi data yang dilakukan dengan menggunakan *Adaptive Huffman*.



(a) (b)

Gambar 2. Proses Kompresi pada Adaptive Huffman: (a) Alur Proses Adaptive Huffman, (b) Prosedur Update

6.2 Pembentukan *Codebook*

Pada proses pembentukan codebook, citra *cover* dibagi menjadi blok-blok berukuran 4 x 4 piksel yang kemudian digunakan sebagai *training vector* [8]. *Initial codebook* (CB_0) dibangitkan dengan mengambil sejumlah vektor (sesuai ukuran *codebook*) dari *training vector* secara *random*. Proses berikutnya yang dilakukan adalah pelatihan *vector* atau *vector training*. Berikut langkah-langkah yang dilakukan pada proses pelatihan vektor:

1. *Training vector* yang memiliki kedekatan dengan *codeword* yang sama (dihitung dengan menggunakan *Euclidian distance*), dikelompokkan ke dalam satu *cluster*.
2. Nilai rataan setiap *cluster* (*centroid*) menggantikan nilai *codeword* tersebut pada *codebook* yang baru CB_{i+1}.
3. Jika *codebook* baru yang terbentuk, $CB_{i+1} \neq CB_i$, maka kembali ke langkah 1.

Pelatihan vektor dilakukan hingga vektor konvergen, $CB_{i+1} = CB_i$.

6.3 Pembentukan Graf

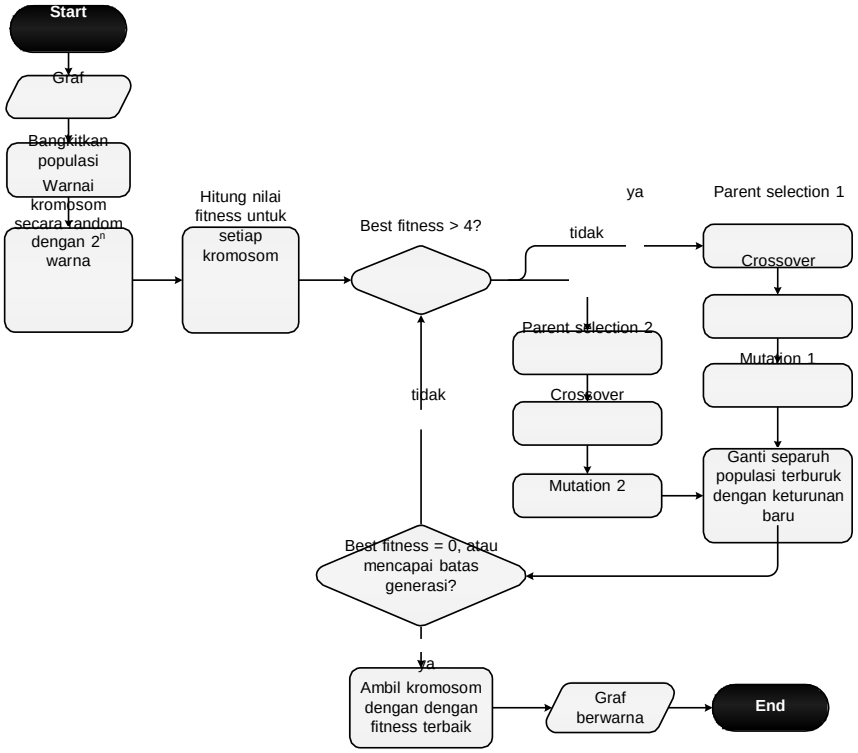
Setelah *codebook* diperoleh, proses yang dilakukan berikutnya adalah pembentukan graf. Setiap *codeword* pada *codebook* direpresentasikan sebagai *vertex* pada graf. Jarak antar *vertex* dihitung berdasarkan *Euclidian distance* dan diantara dua *vertex* diberi busur jika kedua *vertex* memiliki jarak ketetanggaan dalam batas *threshold*, $D(cw_1, cw_2) \leq adj\_tresh$.

6.4 Pewarnaan Graf dengan GA

Graf bertetangga yang dihasilkan pada proses pembentukan graf kemudian diwarnai dengan menggunakan GA. Proses ini diawali dengan pembangkitan populasi berisi sejumlah kromosom. Setiap kromosom terdiri dari gen-gen yang merepresentasikan warna dari setiap *vertex* pada graf bertetangga. Pewarnaan graf kemudian dilakukan dengan menggunakan 2^n warna, dimana dua *vertex* yang bertetangga akan diberi warna yang berbeda.

Genetic Algorithm melakukan pewarnaan dengan memanfaatkan operator *parent selection*, *crossover*, dan *mutation*. Skema penyembunyian data yang digunakan pada penelitian ini menggunakan skema pewarnaan GA yang dilakukan oleh Musa M. Hindi [3], dimana terdapat dua operator *parent selection* dan dua *mutation*. *ParentSelection1* dan *Mutation1* digunakan jika *fitness* terbaik bernilai > 4, sedangkan jika nilai *fitness* yang terbaik bernilai ≤ 4, maka dipilih *ParentSelection2* dan *Mutation2*.

Langkah-langkah yang dilakukan pada proses pewarnaan graf dengan GA dapat dilihat pada gambar berikut:



Gambar 3.Proses Pewarnaan dengan GA

Bad edge (sisi yang buruk) didefinisikan sebagai sisi yang menghubungkan dua *vertex* bertetangga yang memiliki warna sama. Nilai *fitness* untuk suatu kromosom adalah jumlah *bad edge*. Proses pewarnaan dilakukan

hingga tidak ada *vertex* bertetangga yang memiliki warna yang sama, yaitu ketika ditemukan kromosom dengan nilai *fitness* = 0 atau batas generasi terpenuhi.

6.4.1 *Parent Selection*

Seleksi orang tua menggunakan *Parent Selection 1* dilakukan dengan memilih kromosom yang lebih baik diantara 2 kromosom yang terpilih secara *random*. Sedangkan seleksi orang tua dengan menggunakan *Parent Selection 2* dilakukan dengan memilih 2 kromosom yang terbaik pada populasi, yaitu kromosom yang memiliki nilai *fitness* terkecil. Seleksi dilakukan sebanyak jumlah kromosom yang akan digantikan.

6.4.2 *Crossover*

Parents yang dihasilkan pada proses *Parent Selection* menjadi input dari proses berikutnya, yaitu *crossover*. *Crossover* adalah penukaran gen atar dua *parents*. *Crossover* menghasilkan dua *child* (anak) yang mewarisi gen dari *parent1* dan *parent2*. *Parents* yang dihasilkan oleh *Parent Selection 1* dan *Parent Selection 2* akan melalui proses *crossover* yang sama.

Penukaran gen dilakukan berdasarkan *crosspoint* atau titik potong yang ditentukan secara *random*.
child1 = warna dari gen pertama sampai *crosspoint* pada *parent1* + warna dari *crosspoint* sampai warna gen terakhir pada *parent2*.
child2 = warna dari gen pertama sampai *crosspoint* pada *parent2* + warna dari *crosspoint* sampai warna gen terakhir pada *parent1*.

6.4.3 *Mutation*

Mutasi dilakukan untuk memastikan pemberian warna yang berbeda untuk setiap *vertex* yang bertetangga. *Vertex* direpresentasikan sebagai gen dari sebuah kromosom. Setiap gen dari kromosom yang telah melalui proses *crossover* akan dicek warnanya. Pada *Mutation 1*, jika gen yang bertetangga memiliki warna yang sama, maka warna gen tersebut diganti dengan warna lain yang tidak digunakan oleh gen-gen tetangganya. Sedangkan pada *Mutation 2*, jika ditemukan ada gen yang bertetangga memiliki warna yang sama, maka warna gen tersebut akan diwarnai ulang dengan warna yang diambil secara *random* dari warna yang ada di *color_number*.

6.5 *Refine Graph*

Setelah graf diwarnai, graf di-*refine*. *Vertex v* akan dihapus jika verteks tersebut tidak terhubung ke semua warna yang digunakan untuk pewarnaan.

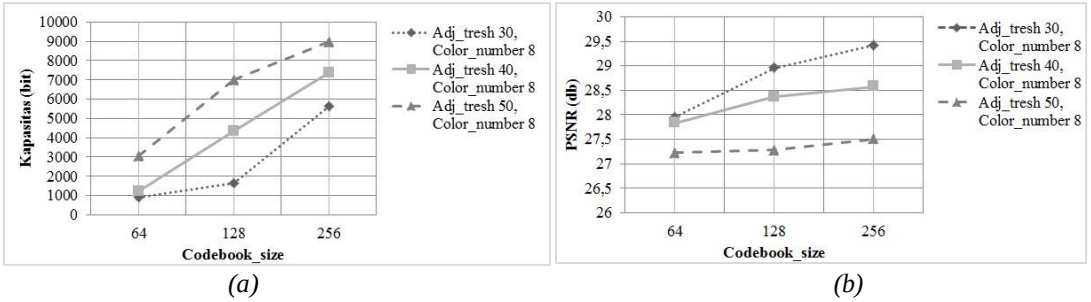
6.6 Penyisipan Bit Data

Pada tahap penyisipan, citra dibagi ke dalam blok-blok. Untuk setiap blok, dicari *codeword* terdekat berdasarkan nilai *Euclidian distance*, kemudian dicek apakah ada *vertex* pada *refined graph* yang berkorespondensi dengan *codeword* tersebut. Jika *vertex* tidak ditemukan, maka kode kompresi untuk blok tersebut = indeks *codeword* terdekat dan tidak ada bit data yang disisipkan pada blok tersebut. Sedangkan jika *vertex* ditemukan, maka skema akan mencari di sekitar *vertex* tersebut, *vertex* yang memiliki warna yang sesuai dengan bit data yang disisipkan, dan kode kompresi = indeks *codeword* yang direpresentasikan oleh *vertex* dengan warna yang berkesesuaian dengan bit data yang disisipkan.

7. Hasil Pengujian dan Analisis

7.1 Pengaruh Parameter *Codebook_size*

Pengujian terhadap pengaruh ukuran *codebook* dilakukan dengan menggunakan ukuran *codebook* yang berbeda pada batas jarak ketetanggaan (*adj_tresh*) dan pemakaian jumlah warna (*color_number*) yang sama. Nilai *codebook_size* yang digunakan adalah 64, 128, dan 256, yaitu penggunaan *codebook* berisi 64, 128, dan 256 *codeword*.



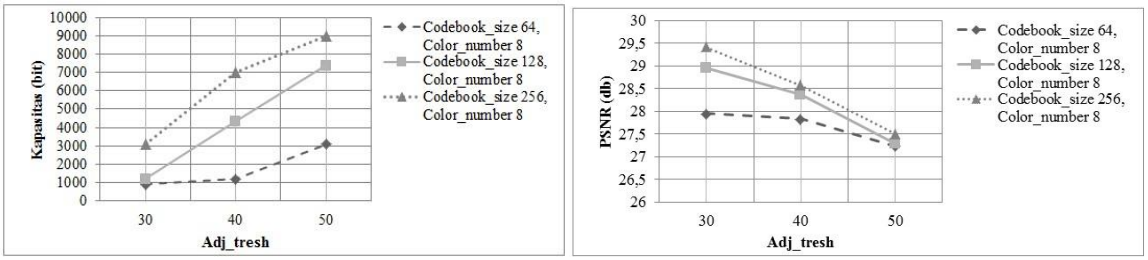
(a) *(b)*

Gambar 4. Pengaruh Parameter Codebook_size Terhadap: (a) Kapasitas Penyisipan dan (b) Nilai PSNR

Berdasarkan hasil pengujian yang dilakukan, seperti ditunjukkan Gambar 4.a di atas, penggunaan ukuran *codebook* yang lebih besar dapat meningkatkan kapasitas penyisipan data. Semakin besar ukuran *codebook*, semakin banyak verteks yang terdapat pada graf, sehingga *refined graph* berpeluang memiliki verteks yang lebih banyak dibandingkan ketika menggunakan ukuran *codebook* yang lebih kecil. Semakin banyak verteks yang terdapat pada *refined graph*, semakin banyak pula verteks yang dapat digunakan untuk penyisipan data. Dengan kata lain, *refined graph* yang memiliki jumlah verteks yang lebih banyak akan menghasilkan kapasitas penyisipan yang lebih besar. Adapun Gambar 4.b menunjukkan bahwa kualitas citra hasil penyisipan juga meningkat pada penggunaan ukuran *codebook* yang lebih besar. Hal ini dapat dilihat dengan meningkatnya nilai PSNR ketika nilai parameter *codebook_size* diperbesar. *Codebook* dengan ukuran yang lebih besar memiliki referensi *codeword* yang lebih banyak, sehingga kualitas citra hasil penyisipan menjadi lebih baik.

7.2 Pengaruh Parameter *Adj_tresh*

Parameter *adj_tresh* (*adjacent threshold*) menunjukkan jarak ketetanggaan antar *vertex*, dimana dua buah *vertex* dikatakan bertetangga jika nilai *Euclidian distance* dua *vertex* tersebut tidak lebih dari *adj_tresh*. Pengujian terhadap pengaruh nilai *adj_tresh* dilakukan dengan mengubah nilai parameter ini pada penggunaan *codebook_size* dan *color_number* yang sama. Nilai *adj_tresh* yang digunakan adalah 30, 40, dan 50.



Gambar 5. Pengaruh Parameter Adj_tresh Terhadap Kapasitas Penyisipan dan Nilai PSNR

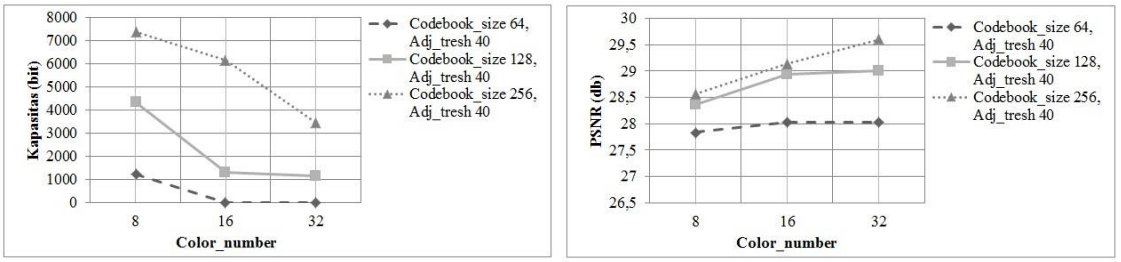
Hasil pengujian menunjukkan secara signifikan bahwa parameter *adj_tresh* ini berpengaruh besar terhadap kapasitas penyisipan. Semakin besar nilai *adj_tresh* yang digunakan, semakin besar pula kapasitas penyisipan yang dihasilkan.

Parameter *adj_tresh* berperan pada pembentukan graf bertetangga. Dua verteks yang memiliki jarak ketetanggaan ≤ *adj_tresh* akan dihubungkan dengan sebuah busur. Semakin besar nilai *adj_tresh*, maka akan semakin banyak verteks yang terkoneksi dan semakin sedikit verteks yang akan dihapus pada saat *refine graph* dilakukan. Dengan demikian, semakin banyak pula verteks yang dapat digunakan untuk penyisipan bit data, sehingga kapasitas penyisipan menjadi lebih besar.

Berbanding terbalik dengan kapasitas penyisipan, nilai PSNR menunjukan penurunan dengan semakin besarnya nilai *adj_tresh* yang digunakan. Semakin banyak penyisipan dilakukan, distorsi citra hasil penyisipan pun semakin besar. Hal ini menyebabkan penurunan nilai PSNR dan kualitas citra hasil penyisipan.

7.3 Pengaruh Parameter *Color_number*

Pengaruh penggunaan jumlah warna untuk pewarnaan yang direpresentasikan oleh nilai parameter *color_number* dilakukan dengan mengubah nilai parameter ini pada penggunaan ukuran *codebook* (*codebook_size*) dan batas ketetanggaan (*adj_tresh*) yang sama. Jumlah warna yang digunakan pada pengujian ini adalah 8, 16, dan 32.



Gambar 6. Pengaruh Parameter Color_number Terhadap Kapasitas Penyisipan dan Nilai PSNR

Kapasitas penyisipan erat kaitannya dengan *refined graph*. Semakin banyak verteks pada *refined graph*, semakin besar pula kapasitas penyisipan yang dapat dihasilkan. Kesesuaian penggunaan warna dengan jarak

ketetanggaan verteks dan ukuran *codebook* memiliki pengaruh yang besar terhadap *refined graph* yang dihasilkan. Misalnya, pada penggunaan *color_number* 8, kapasitas penyisipan meningkat secara stabil seiring dengan meningkatnya nilai *adj_tresh* dan *codebook_size*, karena cukup mudah untuk mendapatkan verteks yang terhubung dengan ke-8 warna tersebut (jumlah warna relatif sedikit), sehingga dapat dihasilkan *refined graph* dengan jumlah verteks yang cukup banyak. Sedangkan skenario penyisipan dengan *color_number* 16 atau 32 dapat menghasilkan kapasitas penyisipan yang lebih kecil atau bahkan menghasilkan kapasitas = 0 bit. Hal ini bergantung pada nilai *adj_tresh* dan *codebook_size* yang digunakan pada penggunaan jumlah warna tersebut. Penggunaan jumlah warna yang lebih banyak perlu diimbangi dengan *adj_tresh* dan *codebook_size* yang lebih besar.

Ketika penyisipan dilakukan dengan menggunakan *color_number* 16 atau 32, verteks yang terhubung dengan semua warna relatif sulit ditemukan, sehingga jumlah verteks pada *refined graph* cenderung lebih sedikit dibandingkan dengan penyisipan yang dilakukan dengan *color_number* 8. Bahkan dapat terjadi kapasitas penyisipan = 0, yaitu ketika tidak ditemukan verteks pada graf yang terhubung dengan semua warna. Adapun nilai PSNR semakin meningkat dengan penambahan jumlah warna yang digunakan, seiring dengan penurunan kapasitas penyisipan yang dihasilkan.

7.4 Hasil Pengujian Terhadap Pengaruh Kompresi

Pengujian terhadap pengaruh kompresi dilakukan dengan melakukan penyisipan dengan kondisi data terkompresi dan tidak terkompresi pada penggunaan parameter *codebook_size*, *adj_tresh*, dan *color_number* yang sama. Adapun hasil pengujian terhadap pengaruh kompresi ditampilkan pada tabel di bawah ini.

Tabel 1 Hasil Pengujian Terhadap Pengaruh Kompresi pada Penyisipan Data Teks

Skenario	Codebook size	Adj tresh	Color number	Kapasitas		PSNR (db)
				(bit)	(karakter)	
Dengan kompresi						
1	64	40	8	1362	272	27,6993
2	128	40	8	4329	865	28,1853
3	256	40	8	7602	1520	28,3888
Tanpa kompresi						
4	64	40	8	1437	179	27,6791
5	128	40	8	4368	546	28,2192
6	256	40	8	7632	954	28,3887

Berdasarkan hasil pengujian yang dilakukan, penyisipan yang dilakukan pada teks terkompresi mengkasilkan kapasitas penyisipan yang lebih besar. Setiap karakter pada teks yang tidak terkompresi direpresentasikan dalam 8 bit *binary*, sedangkan pada teks terkompresi dengan *Adaptive Huffman*, setiap karakter rata-rata direpresentasikan dalam 5 bit *binary*. Dengan demikian, penggunaan kompresi pada data yang akan disisipkan dapat meningkatkan kapasitas penyisipan.

7.5 Hasil Pengujian Terhadap Waktu Pemrosesan

Pengujian terhadap waktu proses dilakukan dengan melihat waktu yang dibutuhkan sistem untuk melakukan pemrosesan pada tahap penyisipan data. Hasil pengujian ini dapat menunjukkan proses apa yang memerlukan waktu pemrosesan yang besar. Berikut ditampilkan pada Tabel 2, hasil pengujian terhadap waktu proses pada tahap penyisipan.

Tabel 2 Hasil Pengujian Terhadap Waktu Proses pada Tahap Penyisipan

CB Size	Adj Tresh	Color	Waktu Pemrosesan (s)						Total Waktu Proses (s)
			Generate Codebook	Generate Graph	Coloring Graph GA	Refine Graph	Compress AddHuff	Embed	
Best fitness = 0									
64	30	8	11,1400	0,0062	0,1400	0,0573	43,8327	4,3944	59,5706
128	40	16	25,0499	0,0128	1,1781	0,1162	43,5995	12,2753	82,2318
256	50	32	45,6595	0,0555	10,2052	0,3250	42,0397	47,8825	146,1674
Best fitness > 0									
256	30	8	46,9317	0,0559	746,5033	0,1003	42,3326	10,6281	846,5519
256	40	8	43,0224	0,0514	2640,6293	0,1027	42,9451	16,4170	2743,1679
256	50	8	35,1490	0,0585	4152,6700	0,1054	43,4516	19,3411	4250,7756

Berdasarkan hasil pengujian yang dilakukan, ukuran *codebook*, jarak ketetanggaan *vertex*, dan penentuan jumlah warna memiliki pengaruh besar terhadap waktu yang dibutuhkan pada proses pewarnaan dengan GA. Pewarnaan graf dengan GA akan semakin mudah dilakukan jika penggunaan jumlah warna diimbangi dengan ukuran *codebook* dan *threshold* jarak ketetanggaan *vertex*. Semakin besar ukuran *codebook* dan *threshold* ketetanggaan *vertex*, semakin banyak pula *vertex* yang terkoneksi. Pada kondisi demikian, jika pewarnaan dilakukan dengan menggunakan jumlah warna yang kecil, semakin lama waktu yang dibutuhkan GA untuk menghasilkan graf dengan *fitness* = 0. Semakin banyak *bad-edge* yang ditemukan pada graf, maka semakin banyak *vertex* bertetangga yang perlu dicek dan diganti warnanya (pada proses mutasi), sehingga semakin lama pula waktu yang dibutuhkan untuk melakukan pewarnaan pada setiap generasi GA. Jika GA tidak berhasil mendapatkan graf dengan nilai *fitness* = 0 (*zero bad-edge*), maka proses pewarnaan berhenti ketika batas generasi dipenuhi dan memerlukan waktu yang lama.

8. Performansi Hasil Penyisipan

Adaptive Huffman menghasilkan rasio kompresi rata-rata 44% dengan rata-rata setiap simbolnya dapat direpresentasikan dalam 5 bit *binary*. Penyisipan data terkompresi dengan *Adaptive Huffman* menggunakan skema yang diajukan pada penelitian ini menghasilkan kapasitas penyisipan terbesar pada *codebook_size* 256, *adj_tresh* 50, dan *color_number* 8, dengan data yang dapat disisipkan sebanyak 9000 bit atau sekitar 1800 karakter, dengan nilai PSNR 27,5054 db. Sedangkan nilai PSNR tertinggi diperoleh pada penggunaan *codebook_size* 256, *adj_tresh* 40, dan *color_number* 32, yaitu PSNR 29,6016 db dengan jumlah bit data yang dapat disisipkan sebanyak 3445 bit atau sekitar 689 karakter.

(a) Citra Asli

(b) Citra Terkompresi VQ
Codebook_size 256
PSNR 30,2487

(c) Citra Tersisipi Data
Codebook_size 256
Adj_tresh: 50
Color_number: 8

9000 bit (sekitar 1800 karakter)
PSNR: 27,5054 db

(d) Citra Tersisipi Data
Codebook_size: 256
Adj_tresh: 40
Color_number: 32

3445 bit (sekitar 689 karakter)
PSNR: 29,6016 db

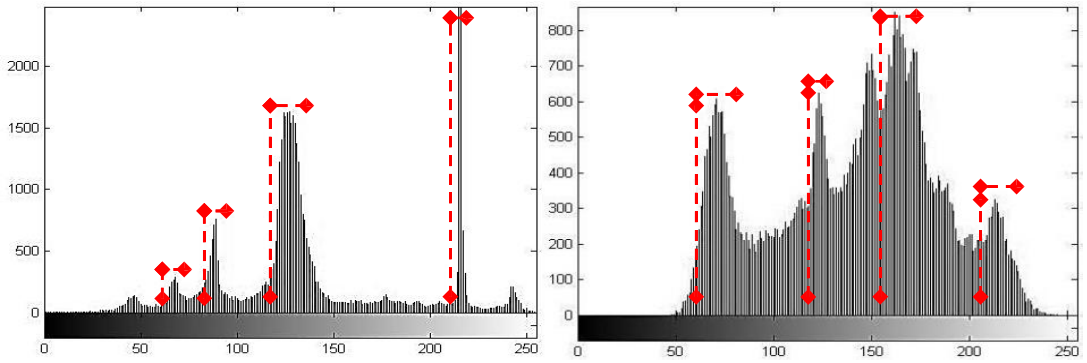
Gambar 7. Perbandingan Citra Sebelum dan Setelah Penyisisipan Dilakukan

Citra hasil kompresi VQ yang memiliki kualitas baik dapat menghasilkan kapasitas dan kualitas citra yang baik setelah penyisipan data dilakukan. Adapun penurunan nilai PSNR sebelum dan setelah penyisipan sebanding dengan jumlah bit data yang tersisipkan. Semakin banyak penyisipan dilakukan, nilai PSNR semakin menurun. Berikut ditampilkan pada Tabel 3, hasil penyisipan pada beberapa citra standar lain yang biasa digunakan pada pengolahan citra dengan nilai parameter *codebook_size* 256, *adj_tresh* 50, dan *color_number* 8.

Tabel 3 Hasil Penyisipan pada Beberapa Citra Standar Pengolahan Citra

No.	Citra	Kompresi VQ	Setelah Penyisipan	
		PSNR (db)	Kapasitas (bit)	PSNR (db)
1	House.jpg	32,4539	8808	30,3937
2	Lena.jpg	30,2487	9000	27,5054
3	Peppers.jpg	28,6165	8076	26,4668
4	Tiffany.jpg	28,1017	9186	25,7765
5	Pirate.jpg	28,0804	7539	26,2373
6	Airplane.jpg	28,0212	7746	26,7370
7	Livingroom.jpg	27,6307	6525	26,2584
8	Boat.jpg	27,5177	7524	25,8690
9	Lake.jpg	26,3922	6576	25,4671
10	Bridge.jpg	25,7773	2619	25,3370
11	Baboon.jpg	24,9500	5199	24,1825

Penyisipan pada karakter citra *cover* yang berbeda juga memberikan hasil penyisipan yang berbeda-beda. Citra dengan karakter piksel yang cenderung homogen (memiliki frekuensi tinggi pada derajat keabuan tertentu) menghasilkan kapasitas dan kualitas hasil penyisipan yang lebih baik dibandingkan citra dengan distribusi derajat keabuan yang lebih merata (*uniform*). Semakin homogen karakter piksel citra, hasil penyisipan yang diperoleh semakin baik. Gambar di bawah ini menunjukkan histogram citra house.jpg yang memiliki penyebaran derajat keabuan yang relatif homogen dan baboon.jpg yang memiliki penyebaran derajat keabuan yang lebih merata.



(a) *(b)*

Gambar 8 Histogram Citra: (a) House.jpg, (b) Lena.jpg

Citra dengan derajat keabuan yang relatif homogen menghasilkan kualitas kompresi VQ dan hasil penyisipan yang lebih baik. Dengan demikian, selain penggunaan nilai parameter *codebook_size*, *adj_tresh*, dan *color_number* yang sesuai, pemilihan citra *cover* sebagai media penyisipan juga berperan dalam menentukan kualitas hasil penyisipan.

9. Analisis Kelemahan Sistem

Berdasarkan performansi yang dihasilkan oleh sistem, beberapa kelemahan pada sistem ini, diantaranya:

a. Perubahan pada citra setelah dilakukan penyisipan masih terlihat secara kasat mata.
b. Proses pewarnaan dengan GA memerlukan waktu yang lama jika tidak diperoleh kromosom dengan nilai *fitness* = 0.

10. Kesimpulan dan Saran

10.1 Kesimpulan

Berdasarkan analisis terhadap sistem yang dirancang dan hasil pengujian yang dilakukan pada penelitian ini, beberapa kesimpulan yang dapat diambil adalah sebagai berikut:

a. Skema ini dapat menyisipkan data sebanyak 9000 bit atau sekitar 1800 karakter, dengan nilai PSNR 27,5054 db.
b. Semakin besar ukuran *codebook* dan jarak ketetanggaan yang digunakan, semakin besar pula kapasitas penyisipan yang dihasilkan.
c. Penggunaan jumlah warna perlu disesuaikan dengan ukuran *codebook* dan jarak ketetanggaan antar *vertex* untuk menghasilkan kapasitas penyisipan yang baik.
d. Besarnya kapasitas penyisipan yang dihasilkan berbanding terbalik dengan nilai PSNR. Semakin besar kapasitas penyisipan yang dihasilkan, nilai PSNR yang diperoleh semakin kecil. Dengan kata lain, semakin banyak data yang disisipkan, kualitas citra yang tersisipi data menjadi semakin rendah.
e. Selain penggunaan nilai parameter *codebook_size*, *adj_tresh*, dan *color_number* yang sesuai, pemilihan citra *cover* sebagai media penyisipan juga berperan dalam menentukan kualitas hasil penyisipan. Citra dengan penyebaran derajat keabuan yang relatif homogen memberikan hasil penyisipan yang lebih baik, ditinjau dari segi kapasitas dan nilai PSNR citra setelah penyisipan dilakukan.

10.2 Saran

Pengembangan yang dapat dilakukan pada tugas akhir ini antara lain:

1. Menggunakan metode pembentukkan *codebook* lain yang dapat menghasilkan *codebook* dengan kualitas yang lebih baik.
2. Menggunakan metode pewarnaan graf lain dengan waktu komputasi yang lebih kecil.

Daftar Pustaka

[1] Adiwijaya, W.A.B Wirayuda, S.D. Winanjuar, dan U. Muslimah. 2012. "The Multiple Watermarking on Digital Medical Image for Mobility and Authenticity." *Operations Research Proceedings.* Springerlink. 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*. 263-268.

[3] Hindi, Musa M. dan Roman V. Yampolskiy. 2012. "Genetic Algorithm Applied to Graph Coloring Problem." *Proceedings of The Twenty Third Midwest Artificial Intelligence and Cognitive Science Conference*. Cincinnati, Ohio: Omnipress – Madison, WISCONSIN. 60-66.

[4] Kaur, Er. Jaspreet dan Er. Karmjeet Kaur. 2012. "Digital Watermark: A Study." *International Journal of Advance Researce in Computer Science and Software Engineering* 2, no. 8: 159-163.

[5] Lu, Tzu-Chuen dan Chin-Yun Chang. 2010. "A Survey of VQ Codebook Generation." *Journal of Information Hiding and Multimedia Signal Processing* 1, no. 3: 190-203.

[6] Sayood, Khalid. 2006. *Introduction to Data Compression Third Edition.* San Francisco: Morgan Kaufmann, Elsevier.

[7] Sumanthi, C.P (dkk). 2013. "A Study of Various Steganography Techniques Used for Information Hiding." *International Journal of Computer Science & Engineering Survey [IJCSIS]* 4, no. 6.

[8] 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.