IMPLEMENTASI SEGMENTASI CITRA DENGAN METODE *FUZZY CO-CLUSTERING* DAN *PARTICLE SWARM OPTIMIZATION* PADA RUANG WARNA CIELAB

Nama	: Munawaroh
NRP	: 5110100160
Jurusan	: Teknik Informatika – FTIf ITS
Dosen Pembimbing I	: Dr.Chastine Fatichah, S.Kom., M.Kom.
Dosen Pembimbing II	: Rully Soelaiman, S.Kom., M.Kom.

Abstrak

Segmentasi citra merupakan suatu metode partisi terhadap citra menjadi beberapa bagian yang homogen berdasarkan kemiripan tertentu. Proses segmentasi sangat penting karena hasil segmentasi mempengaruhi hasil dari proses yang akan dilakukan selanjutnya seperti pengenalan pola. Dalam melakukan segmentasi, telah banyak metode yang digunakan diantaranya adalah segmentasi dengan histogram, segmentasi dengan deteksi tepi, segmentasi dengan Fuzzy C-Means (FCM) dan masih banyak lagi metode lainnya.

Pada tugas akhir ini, metode yang akan diimplentasikan dalam proses segmentasi citra berwarna adalah algoritma Fuzzy Co-Clustering dengan dual fuzzy yaitu dengan dua fungsi keanggotaan yaitu objek dan fitur. Algoritma Fuzzy Co-Clustering For Images (FCCI) dikembangkan dengan menggabungkan jarak antara setiap fitur titik data dengan fitur cluster center sebagai ukuran ketidakmiripan (dissimilarity) dan entropy dari objek dan fitur sebagai kondisi regularisasi dalam fungsi objektif. Untuk mengoptimalkan threshold dalam proses segementasi citra ini digunakan juga algoritma Particle Swarm Optimization dengan modifikasi perilaku bakteri. Dengan adanya algoritma ini hasil segmentasi diharapkan lebih optimal.

IMPLEMENTATION OF FUZZY CO-CLUSTERING AND PARTICLE SWARM OPTIMIZATION FOR IMAGE SEGMENTATION IN CIELAB COLOR SPACE

Name	: Munawaroh
NRP	: 5110100160
Department	: Teknik Informatika – FTIf ITS
Supervisor I	: Dr.Chastine Fatichah, S.Kom., M.Kom.
Supervisor II	: Rully Soelaiman, S.Kom., M.Kom.

Abstract

Image segmentation is a method of the digital image processing to divide an image into some homogeneous regions based on similarity criteria. Segmentation process is very important due to its result gives a significant effect for the next step such as pattern recognition. Many methods have been implemented for segmentation process such as segmentation based on histogram, segmentation using edge detection, segmentation using Fuzzy C-Means (FCM) and so forth.

On this final project,the method which is going to be implemented for image segmentation process is Fuzzy Co-Clustering algorithm with dual fuzzy membership functions. Fuzzy Co-Clustering Algorithm For Images (FCCI) is developed by using combination of distance for every data point features and feature cluster center as a measure of dissimilarity.For condition regularization of objective function, we use entropy of object and feature.Beside that, we also use Particle Swarm Optimization with modification of bacterial behavior for threshold optimization. By using this heuristic algorithm, the result of segmentation is expected to be more optimal.

DAFTAR KODE SUMBER

(Halaman ini sengaja dikosongkan)

DAFTAR VARIABEL

No.	Variabel	Keterangan
1.	C	Jumlah *cluster* yang digunakan $c = (1,...,C)$
2.	N	Jumlah titik data yang ada dalam citra input $(N = 1,...,N)$
3.	u_{ci}	Notasi yang merepresentasikan keanggotaan objek dari titik data ke-i terhadap *cluster* c
4.	T_u	Parameter bobot yang menunjukkan derajat kefuzzian yang mempengaruhi nilai fungsi keanggotaan objek u_{ci}.
5.	T_v	Parameter bobot yang menunjukkan derajat kefuzzian yang mempengaruhi nilai fungsi keanggotaan fitur v_{cj}.
6.	x_i	Notasi yang menunjukkan piksel atau titik data ke-i.
7.	p_c	Notasi yang menunjukkan pusat *cluster*
8.	$Dist(x_i,p_c)$	*Dissimilarty* yang sebanding dengan akar dari *Euclidean distance* antara piksel x_i dan pusat *cluster* p_c
9.	K	Dimensi dari *feature space* yang berasosiasi dengan setiap titik data. Fitur K ini terdiri atas dua fitur warna yaitu a dan b $(K=a^*,b^*)$

No.	Variabel	Keterangan
10.	X	Himpunan dari N titik data yang berasosiasi dengan citra I yang berukuran $N_1xN_2=N$, $X = \{x_1,x_2,...,x_i,...,x_N\}$
11.	x_{ij}	Notasi yang merepresentasikan titik data ke-i terhadap fitur ke-j
12.	p_{cj}	Himpunan dari fitur *cluster center* antara *centroid* ke-c terhadap fitur ke-j
13.	D_{cij}	Merupakan notasi yang merepresentasikan jarak *Euclidean* antara titik data ke-i terhadap *cluster* ke-c dan terhadap fitur ke-j
14.	$U=\{u_{ci}\}$	Notasi yang merepresentasikan himpunana dari fungsi keanggotaan objek u_{ci}
15.	v_{cj}	Notasi untuk keanggotaan fitur antara *cluster* ke-c terhadap fitur ke-j
16.	$V=\{v_{cj}\}$	Notasi untuk merepresentasikan himpunan fungsi keanggotaan fitur v_{cj}
17.	d_{min}	Merupakan jarak minimal antara *cluster centroids* p_{cj} terhadap fitur j
18.	σ	Variasi maksimum di antara semua *cluster* yang berpengaruh terhadap nilai dari validitas *cluster*
19.	S_{old}	Validitas *cluster* awal
20.	S_{new}	Validitas *cluster* baru sebagai hasil dari proses dari algoritma FCCI

No.	Variabel	Keterangan
21.	λ_i	*Lagrange multiplier* yang diaplikasikan untuk memberikan optimasi terhadap fungsi keanggotaan objek
22.	γ_c	*Lagrange multiplier* yang diaplikasikan untuk memberikan optimasi terhadap fungsi keanggotaan fitur
23.	J	Fungsi objektif algoritma FCCI yang diderivasi terhadap U,V, dan P
24.	ε	Batasan eror maksimum dalam pemodelan algoritma FCCI
25.	τ_{max}	Jumlah iterasi maksimal yang dilakukan dalam proses *fuzzy*. Untuk proses awal τ=1 dan iterasi maksimal τ_{max} = 200
26.	e	Merupakan basis logaritma natural yang bernilai e=2,71

BAB II
SEGMENTASI CITRA MENGGUNAKAN *FUZZY CO-CLUSTERING FOR IMAGES* (FCCI) DALAM RUANG WARNA CIELAB

Pada bab ini akan dibahas teori tentang segmentasi citra, konversi citra dari ruang warna RGB ke dalam ruang warna CIELAB, dan pembahasan mengenai teori FCCI.

2.1 Segmentasi Citra

Segmentasi citra merupakan proses membagi citra digital menjadi beberapa segmen (set piksel). Segmentasi citra digunakan untuk menyederhanakan atau mengubah representasi gambar menjadi sesuatu yang lebih bermakna dan lebih mudah untuk dianalisis. Segmentasi citra biasanya digunakan untuk menemukan objek dan batas-batas dalam sebuah citra [4].

Tujuan dari segmentasi citra adalah untuk menyederhanakan atau mengubah representasi gambar menjadi sesuatu yang lebih bermakna dan lebih mudah untuk dianalisis. Segmentasi biasanya digunakan untuk menemukan objek dan batas-batas dalam gambar. Lebih tepatnya, segmentasi citra adalah proses untuk menempatkan label untuk setiap piksel dalam gambar sehingga piksel dengan label yang sama saling berbagi karakteristik visual tertentu. Hasil dari segmentasi citra adalah seperangkat segmen yang secara kolektif mencakup keseluruhan dari gambar. Setiap piksel dalam suatu daerah yang sama memiliki hubungan dengan beberapa properti karakteristik, seperti warna, intensitas atau tekstur.

Segmentasi citra merupakan salah satu pekerjaan paling sulit dalam pengolahan citra. Akurasi dari segmentasi menentukan kesuksesan atau kegagalan prosedur analisis yang terkomputerisasi. Oleh karena itu, sangat penting untuk meningkatkan akurasi pada segmentasi citra.

Algoritma segmentasi didasarkan pada dua karakteristik nilai derajat kecerahan citra, yaitu *discontinuity* dan *similarity*. Pada kategori *discontinuity* citra dibagi atas dasar perubahan yang mencolok dari derajat kecerahannya. Pada kategori *similarity* didasarkan atas *threshold*, *region splitting*, dan *merging*.

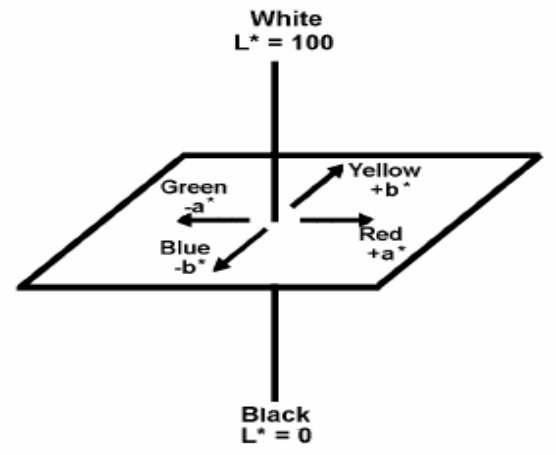
2.2 Metode Konversi Warna

Pada dasarnya, warna terdiri dari tiga dimensi, sehingga disebut *color space*. Contoh dari dimensi warna tersebut adalah RGB, YbCbCr, CIELAB, dan lain-lain. Ruang warna yang sering digunakan adalah RGB, karena memiliki rentang kombinasi yang paling lebar, yaitu 0 sampai dengan 255.

2.2.1 CIELAB *Color Space*

CIELAB merupakan model warna yang dirancang untuk menyerupai persepsi penglihatan manusia dengan menggunakan tiga komponen, yaitu L sebagai *Luminance* (pencahayaan) dan a,b sebagai dimensi warna yang berlawanan [5]. CIELAB dipilih karena terbukti memberikan hasil yang lebih baik dan sensitif daripada warna RGB dalam mengukur kemiripan ciri warna pada citra.

Model warna ini juga dapat digunakan untuk mengatur kontras pencahayaan menjadi lebih baik. Pada Gambar 2.1 yang menunjukkan diagram warna CIELAB dimana L* = koordinat pencahayaan dengan rentang 0 hingga 100, a* = koordinat antara merah dan hijau dengan rentang -127 hingga 127, dan b* = koordinat antara kuning dan biru dengan rentang antara -127 hingga 127.



Gambar 2.1 Diagram Warna CIELAB

2.3 Logika *Fuzzy*

Logika *fuzzy* adalah suatu logika yang memiliki nilai kesamaran antara benar atau salah. Dalam teori logika *fuzzy* suatu nilai bias bernilai benar atau salah secara bersama. Besar keberadaan dan kesalahan sesuatu tergantung pada bobot keanggotaan yang dimilikinya. Logika *fuzzy* memiliki derajat keanggotaan dalam rentang 0 sampai 1. Berbeda dengan logika digital yang memiliki dua nilai, yaitu 0 atau 1.

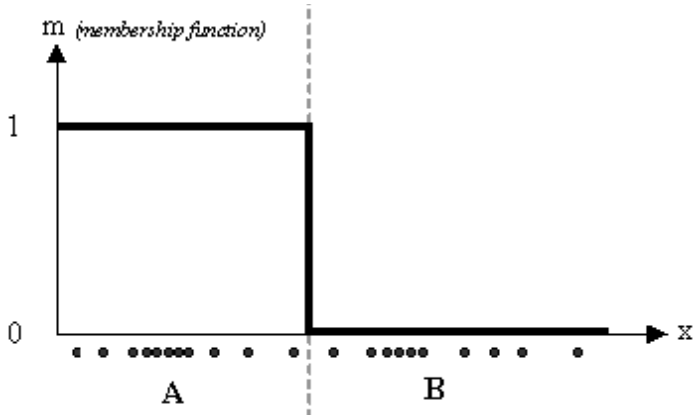
Logika *fuzzy* menunjukan sejauh mana suatu nilai itu benar dan sejauh mana suatu nilai itu salah. Logika *fuzzy* memungkinkan nilai keanggotaan antara 0 dan 1, tingkat keabuan dan juga hitam maupun putih, dan konsep tidak pasti seperti sedikit, lumayan, dan sangat. Logika *fuzzy* pertama kali diperkenalkan oleh Dr. Lotfi Zadeh dari University California, Berkeley pada tahun 1965. Kelebihan dari teori logika *fuzzy* adalah kemampuan dalam proses penalaran secara bahasa

(*linguistic reasoning*). Sehingga dalam perancangannya tidak memerlukan persamaan matematik dari objek yang akan dikendalikan.

2.4 Clustering

Clustering merupakan sebuah teknik pengelompokkan data sesuai dengan *similarity* tertentu ke dalam *cluster* dengan metode tertentu. *Clustering* sering disebut sebagai *unsupervised learning* karena *class* atau *cluster* belum ditentukan. *Clustering* dapat dibedakan ke dalam dua jenis:

1. *Hard clustering*: dimana dalam *clustering* ini, objek atau data *point* hanya menjadi anggota dari satu *cluster* atau tidak. Sehingga tidak memungkinkan sebuah *data point* menjadi anggota lebih dari satu *cluster*.
2. *Soft clustering*: dimana objek atau data *point* menjadi anggota dari sebuah *cluster* atau lebih dengan derajat tertentu.



Gambar 2.2 Pengelompokan Data Secara Tegas

Pada gambar 2.2 menunjukkan bahwa terdapat sebaran piksel, kemudian piksel-piksel tersebut ditentukan masuk ke dalam *cluster* A atau B secara pasti. Sehingga, kemungkinan setiap piksel hanya akan masuk nilai 0 atau 1 yang menunjukkan piksel tersebut masuk *cluster* tersebut atau tidak. *Cluster* seperti ini terdapat pada algoritma *clustering K-Means*.

Sebagian besar *problem* dalam *clustering* berdasarkan pada jarak antar titik data. Dalam hal ini terdapat dua kelas yang menjadi ukuran dalam *clustering*, yaitu:

1. Euclidean Distance

   Euclidean distance merupakan jarak antara dua titik dalam populasi data. Perhitungan *Euclidean distance* ini berdasarkan pada lokasi dari titik-titik dalam ruang dimensi. Tujuan dari *Euclidean distance* ini adalah untuk menentukan jarak antara sebuah data *point* dengan pusat *cluster*.

2. Non-Euclidean

   Untuk jarak *non-euclidean* ini berdasarkan pada properti dari titik data, tetapi bukan yang berkaitan dengan lokasi.

Clustering dikatakan bagus jika memenuhi beberapa kriteria berikut ini:

1. Jarak *intra-cluster* minimal: piksel dalam satu *cluster* memiliki tingkat kesamaan yang tinggi antar piksel yang satu dengan yang lainnya.
2. Jarak *inter-cluster* maksimal: jarak antara *cluster* yang satu dengan yang lain maksimal.

2.5 *Fuzzy Co-Clustering For Images* (FCCI)

Algoritma FCCI dikembangkan dengan menggabungkan jarak antara setiap fitur titik data dan fitur pusat *cluster* sebagai ukuran ketidakmiripan (*dissimilarity*) dan *entropy* dari objek dan fitur sebagai kondisi regularisasi dalam fungsi objektif (*objective*

function). Algoritma ini merupakan pengembangan dari algoritma *Fuzzy C-Means*, hanya saja dalam FCCI ini memiliki dua fungsi keanggotaan yaitu fungsi keanggotaan objek (*object membership function*) dan fungsi keanggotaan fitur (*feature membership function*).

Penggunaan dua fungsi keanggotaan ini bertujuan untuk meminimalisasi masalah *outliers*, dimana masalah ini bisa diminimalisasi dengan menggunakan fungsi keanggotaan fitur. Selain itu dengan hanya menggunakan keanggotaan fitur bisa mengakibatkan *cluster* yang *overlapping*, sehingga dibutuhkan keduanya yaitu fungsi keanggotaan fitur dan fungsi keanggotaan objek.

Konsep dari FCCI ini adalah menentukan jumlah *cluster*, pusat dari *cluster*, dan *Euclidean distance* untuk mendapatkan jumlah *cluster* yang tepat. Masing-masing titik data memiliki derajat keanggotaan untuk masing-masing *cluste*r dan derajat keanggotaan untuk masing-masing fitur. Dengan mengetahui jumlah *cluster* yang tepat, maka bisa didapatkan hasil segmentasi yang mendekati akurat.

Seperti yang telah disebutkan sebelumnya bahwa algoritma FCCI dikembangkan berdasarkan algoritma yang ada sebelumnya yaitu *Fuzzy C-Means* (FCM), dimana fungsi objektif dari FCM ditunjukkan dengan Persamaan 2.1.

$$J_{FCM} = \sum_{c=1}^{C}\sum_{i=1}^{N} u_{ci}\, Dist(x_i, p_c) + T_u \sum_{c=1}^{C}\sum_{i=1}^{N} u_{ci} log u_{ci} \quad (2.1)$$

Dimana *C,N,* merepresentasikan jumlah *cluster* dan titik data secara berurutan, u_{ci} merupakan fungsi keanggotaan *fuzzy*, T_u merupakan faktor bobot dalam *entropy*, sedangkan $Dist(x_i,p_c)$ adalah *Euclidean distance* antara piksel x_i dan pusat *cluster* p_c.

Dari Persamaan 2.1 di atas diberikan konstrain Persamaan 2.2 sebagai *entropy* yang menjalankan peran sebagai faktor regulasi selama proses minimalisasi. Pendekatan ini berguna untuk *Co-clustering* dalam kerangka kerja berbasis *entropy* dalam FCM.

$$\sum_{c=1}^{C} u_{ci} = 1, u_{ci} \in [0,1], \forall i = 1, \dots, N \quad (2.2)$$

Kemudian dalam fungsi objektif dari FCM di atas, $Dist(x_i,p_c)$ diganti dengan $Dist(x_{ij},p_{cj})$. $Dist(x_{ij},p_{cj})$ dihitung untuk setiap fitur j=1,2,..,K secara terpisah. K merupakan dimensi dari fitur yang berasosiasi dengan setiap titik data.

Misalkan $X=\{x_1,x_2,x_3,\dots,x_i,\dots x_N\} \epsilon R^K$ merupakan himpunan dari N titik data yang berasosiasi dengan citra *I* dengan ukuran $N_1 x N_2 = N$, dan *K* adalah dimensi fitur yang berasosiasi dengan setiap titik data. Misal x_{ij} merepresentasikan sebuah fitur ke-j dari titik data ke-i, $P=\{pcj\}$ merupakan sebuah himpunan fitur pusat *cluster* dan $D_{cij}=Dist(xij,pcj)$ merupakan akar *Euclidean* antara fitur titik data x_{ij} dan fitur pusat *cluster* p_{cj}, maka akan didapatkan persamaan jarak *Euclidean* yang ditunjukkan dalam Persamaan 2.3 berikut ini.

$$D_{cij} = d^2(x_{ij}, p_{cj}) = (x_{ij} - p_{cj})^2 \quad (2.3)$$

Kemudian keanggotaan objek dari titik data ke-i terhadap *cluster c* direpresentasikan dengan u_{ci}, dan $U=\{ u_{ci} \}$ merupakan matriks fungsi keanggotaan dari citra *I*. Sedangkan v_{cj} merepresentasikan keanggotaan fitur yang didefinisikan sebagai keanggotaan fitur *j* terhadap *cluster c* dan $V=\{v_{cj}\}$ merupakan matriks keanggotaan fitur dari citra *I*.

Dengan memasukkan fungsi keanggotaan fitur v_{cj} ke dalam Persamaan 2.1 dan mengganti fungsi jarak dengan D_{cij}

maka akan menghasilkan $\sum_{c=1}^{C}\sum_{i=1}^{N}\sum_{j=1}^{K} u_{ci}\, v_{cj} D_{cij}$ yang dianggap sebagai derajat dari agregrasi dari fungsi objektif J_{FCCI}. Pengaturan *entropy* secara terpisah $\sum_{c=1}^{C}\sum_{i=1}^{N} u_{ci} \log u_{ci}$ dan $\sum_{c=1}^{C}\sum_{j=1}^{K} v_{cj} \log v_{cj}$ untuk fungsi keanggotaan objek dan fitur yang nantinya akan menghasilkan rumus kedua dan ketiga dari J_{FCCI}.

Meminimalisasi kedua rumus diatas sebanding dengan memaksimalkan *fuzzy entropy* $-\sum_{c=1}^{C}\sum_{i=1}^{N} u_{ci} \log u_{ci}$ dan $-\sum_{c=1}^{C}\sum_{j=1}^{K} v_{cj} \log v_{cj}$. *Entropy* dimaksimalkan ketika keanggotaan *fuzzy* u_{ci} dan v_{cj} didistribusikan secara *uniform* berdasarkan *constraint* mereka misalkan u_{ci} = 1/C dan v_{cj}=1/K. Dengan menggabungkan semua persamaan di atas, maka dihasilkan fungsi objektif untuk *Fuzzy Co-Clustering For Images* yang ditunjukkan dalam Persamaan 2.4 berikut ini.

$$J_{FCCI}(U,V,P) = \sum_{c=1}^{C}\sum_{i=1}^{N}\sum_{j=1}^{K} u_{ci} v_{cj} D_{cij} + T_U \sum_{c=1}^{C}\sum_{i=1}^{N} u_{ci} log u_{ci} + T_v \sum_{c=1}^{C}\sum_{j=1}^{K} v_{cj} log v_{cj} \quad (2.4)$$

Nilai dari U^*, V^*, dan P^* pada fungsi objektif FCCI di atas masih belum diketahui, fungsi objektif tersebut tidak konkaf ataupun konvek dan biasanya memiliki banyak *local optima*. Sehingga untuk menemukan *global optimum*, digunakan algoritma *Particle Swarm Optimization* (PSO) dengan Modifikasi Perilaku Bakteri (PSO-MPB) untuk mengoptimasikan nilai dari parameter bobot yaitu T_U dan T_V.

Dari persamaan fungsi objektif di atas, terdapat tiga bagian yang tiap bagiannya memberikan makna yang dapat dijelaskan sebagai berikut:

1. $\sum_{c=1}^{C}\sum_{i=1}^{N}\sum_{j=1}^{K} u_{ci}\, v_{cj} D_{cij}$

Dalam *co-clustering* ini, hal yang menentukan keanggotaan sebuah *data point* selain ditentukan oleh fungsi keanggotaan objek u_{ci} , *Euclidean distance* D_{cij} antara sebuah titik data dengan pusat *cluster,* juga ditentukan oleh *feature membership function* v_{cj}. Jika dalam fungsi keanggotaan FCM, besarnya fungsi keanggotaan objek hanya dikalikan dengan jarak paling minimal antara sebuah titik data dengan pusat *cluster*, maka dalam FCCI ini ditambahkan dengan mengalikan ketiganya baik jarak *Euclidean* paling minimal, besar derajat keanggotaan objek maupun besar derajat keanggotaan fitur.

2. $T_U \sum_{c=1}^{C}\sum_{i=1}^{N} u_{ci} \log u_{ci}$

Persamaan di atas merupakan *entropy* yang berguna untuk menentukan keberagaman suatu data dalam fungsi objektif FCCI. Semakin heterogen maka nilai *entropy* akan semakin besar.*Fuzzy co-clustering* disini menggunakan *entropy* diatas dengan tujuan untuk menentukan *purity* dari suatu titik data terhadap kelompoknya. Dari formula di atas dapat dilihat bahwa nilai *entropy* ini dipengaruhi oleh nilai dari derajat keanggotaan objek dan konstanta T_u yang merupakan parameter bobot yang berperan sebagai *threshold* dalam segementasi. Keanggotaan objek u_{ci} ini berperan sebagai p_i yaitu jumlah proporsi sampel yang merupakan peluang untuk *cluster C*. Dalam *entropy* ini, semakin kecil nilai *entropy,* maka data tersebut semakin homogen. Jadi dari formula ini diharapkan nilai *entropy* semakin kecil, sehingga data semakin homogen.

3. $T_V \sum_{c=1}^{C}\sum_{j=1}^{K} v_{cj} \log v_{cj}$

Formula ini sama dengan formula sebelumnya. Hanya saja untuk formula ini nilai yang menentukan besarnya suatu *entropy*

dilihat dari nilai fungsi keanggotaan fiturnya beserta nilai konstanta T_v nya.

Persamaan 2.4 yang menunjukkan fungsi objektif dari *FuzzyCo-Clustering For Images* (FCCI) kemudian diminimalisasi dengan dua konstrain berikut yang ditunjukkan pada Persamaan 2.5 dan Persamaan 2.6.

$$\sum_{c=1}^{C} u_{ci} = 1, u_{ci} \in [0,1], \forall i = 1, \dots, N \quad (2.5)$$

Tujuan dari *clustering* adalah pembagian semua *dataset* ke dalam sejumlah *cluster C* untuk semua titik data *i* dari 1 sampai ke-N. Maka semua *cluster* harus memiliki anggota yaitu titik data *i* dan setiap titik data harus menjadi bagian dari sebuah *cluster* yang ditentukan berdasarkan nilai keanggotaan objeknya .

$$\sum_{j=1}^{K} v_{cj} = 1, v_{cj} \in [0,1], \forall c = 1, \dots, C \quad (2.6)$$

Pembagian semua fitur *j* ke dalam sejumlah *cluster C* untuk semua *cluster* dari *cluster* 1 sampai dengan *cluster* ke-C. Maka semua *cluster* harus memiliki anggota yaitu fitur *j* dan setiap fitur *j* harus menjadi bagian dari sebuah *cluster* yang ditentukan berdasarkan nilai keanggotaan fiturnya .

Untuk mengoptimasikan fungsi objektif FCCI dengan dua konstrain di atas maka digunakan *Lagrange Multiplier* dengan variabel λ_i dan γ_c sehingga didapatkan fungsi objektif sesuai dengan Persamaan 2.7 berikut:

$$J(\mathbf{U},\mathbf{V},\mathbf{P}) = \sum_{c=1}^{C} \sum_{i=1}^{N} \sum_{j=1}^{K} u_{ci}\, v_{cj} D_{cij}$$

$$+ T_U \sum_{c=1}^{C} \sum_{i=1}^{N} u_{ci} \log u_{ci} + T_V \sum_{c=1}^{C} \sum_{j=1}^{K} v_{cj} \log v_{cj} +$$

$$\sum_{i=1}^{N} \lambda_i \ (\sum_{c=1}^{C} u_{ci} - 1) + \sum_{c=1}^{C} \gamma_c \ (\sum_{j=1}^{K} v_{cj} - 1) \qquad (2.7)$$

Dari fungsi objektif tersebut, akan diturunkan lagi sehingga nantinya akan menghasilkan fungsi objektif untuk keanggotaan objek u_{ci}, fungsi objektif untuk keanggotaan fitur v_{cj}, dan pusat *cluster* p_{cj}.

2.5.1 Fungsi Keanggotaan Objek u_{ci}

Untuk mendapatkan fungsi u_{ci}, maka Persamaan 2.7 di atas diturunkan terhadap U yang merupakan himpunan fungsi keanggotaan objek. Sehingga didapatkan fungsi keanggotaan objek untuk FCCI ditunjukkan pada Persamaan 2.8 berikut ini:

$$u_{ci} = \frac{e^{\left(-\sum_{j=1}^{K} \frac{v_{cj} D_{cij}}{T_u}\right)}}{\sum_{c=1}^{C} e^{\left(-\sum_{j=1}^{K} \frac{v_{cj} D_{cij}}{T_u}\right)}} \qquad (2.8)$$

Fungsi keanggotaan objek inilah nantinya yang akan digunakan untuk menghitung nilai dari keanggotaan objek sebuah titik data dalam implementasi dengan kode sumber.

2.5.2 Fungsi Keanggotaan Fitur v_{cj}

Sama halnya dengan cara mendapatkan fungsi keanggotaan objek u_{ci}, untuk mendapatkan fungsi keanggotaan fitur v_{cj}, maka Persamaan 2.7 diturunkan terhadap V yang merupakan himpunana fungsi keanggotaan fitur. Fungsi keanggotaan fitur yang didapatkan, ditunjukkan pada Persamaan 2.9 berikut ini:

$$v_{cj} = \frac{e\left(-\sum_{i=1}^{N} \frac{u_{ci}\, D_{cij}}{T_V}\right)}{\sum_{j=1}^{K} e\left(-\sum_{i=1}^{N} \frac{u_{ci}\, D_{cij}}{T_v}\right)} \quad (2.9)$$

Fungsi keanggotaan fitur ini juga nantinya akan digunakan untuk menghitung nilai dari keanggotaan fitur dari sebuah fitur terhadap *cluster* dalam implementasi dengan kode sumber.

2.5.3 Fungsi Untuk Pusat *Cluster* p_{cj}

Sama halnya dengan cara mendapatkan fungsi keanggotaan objek u_{ci} dan keanggotaan fitur v_{cj}, untuk mendapatkan fungsi pusat *cluster* p_{cj}, maka Persamaan 2.7 diturunkan terhadap P yang merupakan himpunana pusat *cluster*. Fungsi pusat *cluster* yang didapatkan, ditunjukkan pada Persamaan 2.10 berikut ini:

$$p_{cj} = \frac{\sum_{i=1}^{N} u_{ci} x_{ij}}{\sum_{i=1}^{N} u_{ci}} \quad (2.10)$$

Fungsi ini nanti akan digunakan untuk menghitung pusat *cluster* dalam implementasi dengan kode sumber.

2.6 Evaluasi Kualitas *Cluster*

Evaluasi kualitas *cluster* pada FCCI dapat dilakukan dengan berbagai cara, salah satunya dalam tugas akhir ini dengan menggunakan fungsi evaluasi yang diusulkan oleh Xie and Beni [6]. Fungsi evaluasi tersebut berfungsi sebagai tolok ukur kuantitatif yang dapat digunakan untuk mengevaluasi kualitas *cluster*. Fungsi validitas Xie and Beni untuk *worst case* ditunjukkan dengan Persamaan 2.11 berikut:

$$S = \frac{\sigma/_N}{{d_{min}}^2} \quad (2.11)$$

S merupakan fungsi evaluasi *cluster* yang nilainya didapatkan dari hasil algoritma *Fuzzy Co-Clustering For Images* (FCCI) yang telah dijalankan sebelumnya. Kemudian nilai *S* ini akan dibandingkan dengan nilai *S* sebelumnya yang di awal telah didefinisikan oleh dengan nilai *S* = 50, apakah nantinya nilai *S* yang baru akan lebih besar dari *S* yang lama. Jika nilai *S* yang baru lebih kecil, maka akan melakukan proses perulangan FCCI lagi hingga nilai *S* yang baru lebih besar atau sama dengan *S* yang lama.

Nilai d_{min} dari Persamaan 2.11 di atas diperoleh dari Persamaan 2.12 beikut ini:

$$d_{min} = \min_{\forall C}\left\{\sum_{j=1}^{K}\left(p_{(c+1)j} - p_{cj}\right)^2\right\} \quad (2.12)$$

Dimana d_{min} adalah jarak minimal antara pusat *cluster* p_{cj} untuk *cluster* c=*1,...,C* dan fitur *j* dan σ adalah variasi maksimal antara semua *cluster* C, yang didefinisikan dengan Persamaan 2.13 berikut:

$$\sigma = \max_{\forall C}\left\{\sum_{i=1}^{N} u_{ci}{}^2 \sum_{j=1}^{K}(x_{ij} - p_{cj})^2\right\} \quad (2.13)$$

2.7 Evaluasi Kuantitatif Segmentasi Citra

Dalam mengevaluasi hasil segmentasi citra nyata maupun citra yang disintesis, serta mengevaluasi hasil baik lokal maupun global, salah satu fungsi yang dapat digunakan adalah fungsi yang diusulkan oleh Liu dan Yang [7].

Kelebihan dari fungsi Liu dan Yang ini adalah metode ini dapat memberikan pengukuran yang akurat terhadap perbedaan warna yang didapatkan oleh algoritma segmentasi dan sekaligus memberikan penghitungan pada banyaknya region yang terbentuk.

Fungsi penghitungan evaluasi kuantitatif segmentasi citra yang diusulkan oleh Liu dan Yang ini ditunjukkan dengan Persamaan 2.14. Fungsi tersebut menghitung perbedaan antara nilai piksel citra asli dengan piksel citra tersegmentasi. Dari perbedaan itu ditemukan *error color* yang dihasilkan oleh citra tersegmentasi.

$$F(I) = \frac{1}{1000N}\sqrt{G}\sum_{i=1}^{G}\frac{ei^2}{\sqrt{Ai}} \quad (2.14)$$

Dimana *N* adalah jumlah piksel citra. *G* adalah jumlah *cluster,* A_i adalah jumlah piksel dari daerah ke-*i*. e_i^2 adalah jumlah jarak *Euclidean* dari vektor antara citra asli dan masing-masing piksel pada daerah citra segmentasi dan *I* adalah citra yang disegmentasi. Hasil segmentasi citra yang baik ditunjukkan dengan nilai evaluasi kuantitas yang rendah.

BAB III
OPTIMASI PARAMETER *THRESHOLD* DENGAN *PARTICLE SWARM OPTIMIZATION* (PSO)

Pada Bab 3 dijelaskan mengenai optimasi parameter *threshold* yaitu T_u dan T_v yang terdapat dalam fungsi objektif FCCI yang memberikan pengaruh besar dalam proses segmentasi citra dengan FCCI. Optimasi terhadap kedua parameter ini dilakukan dengan menggunakan *Particle Swarm Optimization* dengan Modifikasi Perilaku Bakteri (PSO-MPB). Pertama, akan diberikan penjelasan tentang optimasi, kemudian algoritma PSO dan PSO-MPB itu sendiri.

3.1 Optimasi

Pada bidang matematika dan ilmu komputer, optimasi merupakan pencarian elemen terbaik dari beberapa alternatif yang tersedia. Permasalahan dalam optimasi dapat berupa minimisasi dan maksimisasi sebuah fungsi. Penjelasan lebih lengkap dapat dilihat pada sub bab berikut.

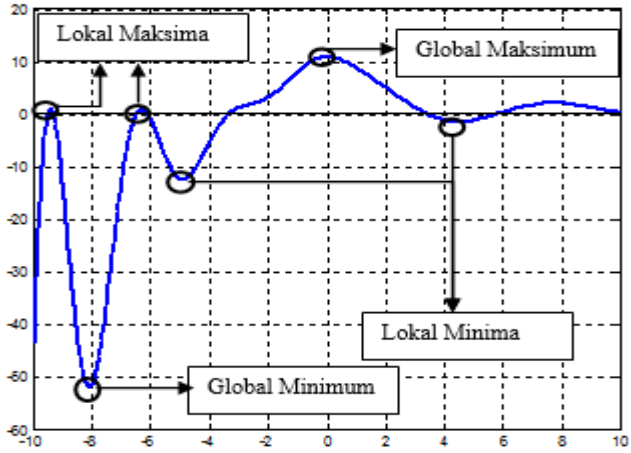
3.1.1 Permasalahan Optimasi

Sebuah permasalahan optimasi dapat direpresentasikan dalam bentuk $f: A \rightarrow R$, dimana A adalah satu set angka real. Apabila salah satu elemen x_0 memiliki nilai $f(x_0) \leq f(x)$ untuk semua x pada A, maka permasalahan tersebut merupakan minimisasi. Tetapi, jika $f(x_0) \geq f(x)$ untuk semua x pada A maka disebut maksimisasi. Domain A untuk nilai f dinamakan *search space* atau satu set solusi. Sedangkan, elemen A disebut kandidat solusi. Fungsi f memiliki arti yang bermacam-macam, antara lain

fungsi objektif, fungsi biaya (minimisasi), atau fungsi utilitas (maksimisasi). Sebuah solusi yang meminimalkan atau memaksimalkan fungsi objektif dinamakan solusi yang optimal.

3.1.2 Optimasi Global

Tugas optimasi global adalah mencari solusi yang memiliki nilai terkecil pada satu set solusi, *global minimum*. Optimasi global tidak hanya menentukan *local minimum*, namun mencari nilai terkecil dalam satu set nilai *local minimum*. Pada sebagian besar permasalahan optimasi *nonlinear*, fungsi objektif *f* mempunyai jumlah *local minimum* dan *local maximum* yang sangat besar. Untuk mendapatkan nilai *local minimum* dapat langsung menggunakan metode optimasi lokal, namun sebaliknya, untuk mendapatkan nilai global minimum dari sebuah fungsi lebih sulit.



Gambar 3.1 Optimasi Global

3.2 *Particle Swarm Optimization* (PSO)

Particle Swarm Optimization adalah sebuah teknik komputasi evolusioner yang dikembangkan oleh Eberheart & Kennedy pada tahun 1995 dan berdasarkan pada kawanan burung [8]. PSO merupakan teknik *meta-heuristic* dimana PSO ini membuat sedikit asumsi tentang masalah yang sedang dioptimasi dan melakukan pencarian pada ruang kandidat solusi yang sangat luas. Selama ini PSO telah terbukti sebagai algoritma optimasi yang paling menjanjikan dalam menyelesaikan masalah yang rumit dalam dunia teknik dan sains. Kesederhanaan dan kecepatannya membuat PSO menjadi algoritma yang menarik untuk diimplementasikan. Dalam PSO populasi disebut kawanan *(swarm)* dan individu diistilahkan sebagai partikel.

Seperti yang telah dijelaskan sebelumnya, PSO menirukan perilaku sosial sekolompok burung. Tiap partikel akan berperilaku sesuai dengan kecerdasan (*intelligent*) dan lingkungan populasinya. Diumpamakan sekelompok burung mencari makanan pada sebuah area. Hanya ada satu makanan pada area tersebut dan tidak ada satu ekor burung yang mengerti lokasi tempat makanan berada. Cara yang paling efektif adalah mengikuti burung yang memiliki posisi paling dekat dengan makanan. Dengan demikian, jika satu partikel atau seekor burung menemukan jalan yang tepat atau pendek menuju sumber makanan, anggota yang lain akan mengikuti jalan tersebut meskipun lokasi mereka jauh dari kelompoknya.

Berdasarkan penjelasan di atas, pemeran utama dalam simulasi ini adalah sekelompok burung yang memiliki tujuan mencari lokasi makanan. Dalam populasi, masing-masing partikel memiliki posisi $x_i = (x_{i1}, x_{i2}, .., x_{iN})$ dan kecepatan $v_i = (v_{i1}, v_{i2}, ..,$

v_{iN}), dimana *i* adalah jumlah partikel dan *N* adalah jumlah dimensi permasalahan atau jumlah variabel yang belum diketahui dari suatu fungsi.

Langkah pertama algoritma PSO yaitu menentukan posisi dan kecepatan awal partikel secara acak. Pada tiap iterasi, nilai masing-masing partikel berubah sesuai dengan dua nilai terbaik. Nilai pertama adalah posisi terbaik tiap partikel yang telah didapat, nilai ini dinamakan pbest. Nilai terbaik yang kedua disebut gbest atau *global best*. Nilai gbest didapatkan dari nilai partikel yang memiliki nilai *fitness* terbaik apabila dibandingkan partikel yang lain. Setelah menemukan dua nilai tersebut, posisi dan kecepatan tiap partikel akan berubah sampai menemukan hasil yang optimal. Perubahan posisi dan kecepatan mengikuti Persamaan 3.1 berikut :

$$v_i^k = wv_i^k + c_1 r_1 (pbest_i^k - x_i^k) + c_2 r_2 (gbest^k - x_i^k) \quad (3.1)$$

$$x_i^{k+1} = x_i^k + v_i^k \quad (3.2)$$

Dimana v_i^k merupakan kecepatan partikel i pada iterasi ke-k, dan x_i^k merupakan posisi partikel i pada iterasi ke-k. Variabel c_1 dan c_2 adalah konstanta positif, dan r_1,r_2 merupakan variabel acak yang terdistribusi *uniform* antara 0 dan 1. Pada persamaan di atas, *w* adalah bobot yang menunjukkan akibat perpindahan kecepatan sebelumnya.

3.3 PSO dengan Modifikasi Perilaku Bakteri (PSO-MPB)

Dalam melakukan optimasi terhadap parameter T_u dan T_v yang ada dalam fungsi objektif FCCI, digunakan algoritma *meta-heuristic* yaitu *Particle Swarm Optimization* (PSO) dengan modifikasi perilaku bakteri. Secara umum, algoritma

PSO itu sendiri merupakan algoritma yang cara kerjanya berdasarkan sebuah kawanan (*swarm*). Kawanan yang dalam algoritma PSO ini biasanya diumpakan dengan kawanan burung, maka dalam tugas akhir ini diganti dengan kawaan bakteri. Karakteristik dari bakteri ini didapatkan berdasarkan algoritma *Bacterial Foraging* [9].

Hal ini dikarenakan jika hanya menggunakan PSO saja yang berorientasi pada *update* posisi dan kecepatan tidak bisa optimal. Sehingga diperlukan karakteristik dari bakteri *E.Coli* yang dapat berorientasi dengan rotasi melalui proses *tumble* dan *swim*. Hal ini telah dibuktikan lebih optimal dibandingkan dengan jika tidak menggunakan modifikasi.

Untuk lebih jelas mengenai algoritma ini, maka disini diberikan penjelasan mengenai algoritma tersebut beserta mekanisme kerjanya.

3.3.1 Karakteristik Bakteri *E.Coli*

Dalam pergerakannya bakteri menggunakan *flagella* sebagai alat gerak. Selama mencari makan dari bakteri pada dunia nyata, daya penggerak didapat dengan satu set *flagella* yang dapat direnggangkan. *Flagella* membantu bakteri *E. coli* untuk berguling dan berenang, dimana ada dua operasi dasar yang dilakukan oleh bakteri pada saat mencari makan [10].

Ketika bakteri memutar *flagela* dalam arah jarum jam, masing-masing menarik *flagela* pada sel yang menghasilkan pergerakan flagella bebas dan akhirnya bakteri beguling dengan jumlah yang lebih kecil jatuh, sedangkan di tempat yang berbahaya sering terjatuh untuk menemukan gradient nutrisi.

Dengan menggunakan kawanan bakteri sebagai pengganti kawanan burung yang ada dalam algoritma PSO ini diharapkan dapat menemukan nilai optimum fungsi objektif FCCI tersebut

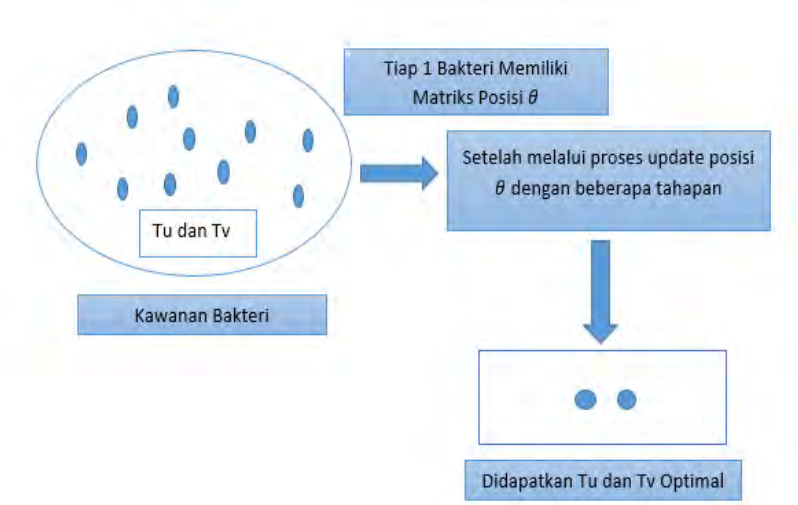
tanpa harus mengetahui *gradient* dari fungsi. Disinilah kelebihan algoritma ini yang dapat menyelesaikan persamaan yang rumit tanpa harus melibatkan *gradient* dari persamaan tersebut.

Dalam algoritma ini nantinya, bakteri memiliki posisi (θ) yang merepresentasikan solusi dari masalah. Pada awalnya θ diinisialisasikan secara *random*. Tiap posisi dari bakteri memiliki *cost* yang berasosiasi dan merepresentasikan *value* dari fungsi yang akan dioptimasi. Satu *bacterium* memiliki posisi yang direpresentasikan dengan $\theta(i,j,k,l)$ yang merupakan matriks empat dimensi. Selain itu sebuah bakteri juga memiliki sebuah fungsi *cost* dari posisinya tadi yang direpresentasikan dengan J(θ).

Dalam tugas akhir ini, parameter yang akan dioptimasi adalah T_u dan T_v, sehingga jika direpresentasikan ke dalam algoritma PSO-MPB ini kedua parameter tersebut akan dianggap sebagai sekawanan bakteri yang sebenarnya merupakan beberapa himpunan solusi dari masalah optimasi. Untuk lebih memberikan pemahaman mengenai teori ini, maka jika dalam optimasi ini inisialisasi bakteri diberikan sejumlah 10 bakteri misalnya maka dalam 10 bakteri tersebut akan terdapat satu T_u dan satu T_v yang optimal.

Nantinya 10 bakteri ini akan direpresentasikan ke dalam bentuk matriks empat dimensi yang merupakan posisi dari bakteri tersebut. Seperti yang disebutkan sebelumnya bahwa tiap bakteri memiliki posisi yang dinyatakan dalam $\theta(i,j,k,l)$. Pemberian inisialisasi posisi ini dilakukan untuk bakteri dari 1 sampai dengan sejumlah bakteri yaitu 10 misalnya. Kemudian, proses selanjutnya adalah mendapatkan posisi awal yang terbaik untuk masing-masing parameter dengan mencari *minimal cost* dari masing-masing posisi bakteri. Setelah itu, bakteri akan mengalami beberapa tahapan hingga didapatkan nilai dua parameter yang

optimal. Representasi dari bakteri untuk proses optimasi ini dapat dilihat pada Gambar 3.2 berikut ini.



Gambar 3.2 Representasi Bakteri dalam Permasalahan Optimasi

3.3.2 Mekanisme Algoritma PSO dengan Modifikasi Perilaku Bakteri (PSO-MPB)

Dalam menjalankan optimasi dengan algoritma ini ada tiga mekanisme dasar yaitu sebagai berikut:

3.3.2.1 Tahapan Inisialisasi dan *Update Fitness Function*

Pada proses optimasi ini, hal pertama yang dilakukan adalah melakukan inisialisasi terhadap parameter-parameter yang akan digunakan. Setelah inisialisasi, maka langkah selanjutnya adalah memberikan inisialisasi posisi untuk bakteri berdasarkan batas atas dan batas bawah yang telah ditentukan. Setelah itu, langkah yang

dilakukan adalah melakukan *update* terhadap *fitness function* untuk masing-masing bakteri dengan berdasarkan Persamaan 3.3 berikut.

$$J(i,j,k,l) = J(i,j,k,l) + J_{cc}(\theta^i(j,k,l) , P(j,k,l)) \quad (3.3)$$

Kemudian, untuk menentukan posisi awal dan cost dari fitness function yang terbaik, maka dilakukan pengecekan terhadap *cost* dari *fitness function* yang telah didapatkan sebelumnya. Jika *cost fitness function* dari bakteri kurang dari *cost optimal function* sebelumnya, maka akan diambil nilai yang lebih kecil tersebut.

3.3.2.2 Tahapan Kemotaksis

Proses ini mensimulasikan gerakan dari sel *E.coli* melalui *swimming* dan *tumble* dengan *flagela*. Secara biologi sebuah bakteri *E. coli* dapat bergerak dalam dua cara yang berbeda, yaitu dapat *swim* untuk jangka waktu dalam arah yang sama atau *tumble*, dan alternatif antara kedua cara untuk seumur hidup.

Pada kemotaksis, komputasi untuk pergerakan bakteri dapat diwakili oleh Persamaan 3.4 berikut:

$$\theta^i(j+1,k,l) = \theta^i(j,k,l) + C(i)\frac{\Delta i}{\sqrt{\Delta^T(i)\Delta(i)}} \quad (3.4)$$

dimana Δ mengindikasikan vektor dalam arah random yang elemennya berada antara [-1,1].

Misalkan $\theta^i(j,k,l)$ merupakan representasi dari bakteri ke-i pada kemotaksis ke-j, reproduksi ke-k, dan eliminasi dispersal ke-l, maka untuk mendapatkan θ^i berikutnya langkah yang dilakukan adalah dengan menambahkan θ^i sebelumnya

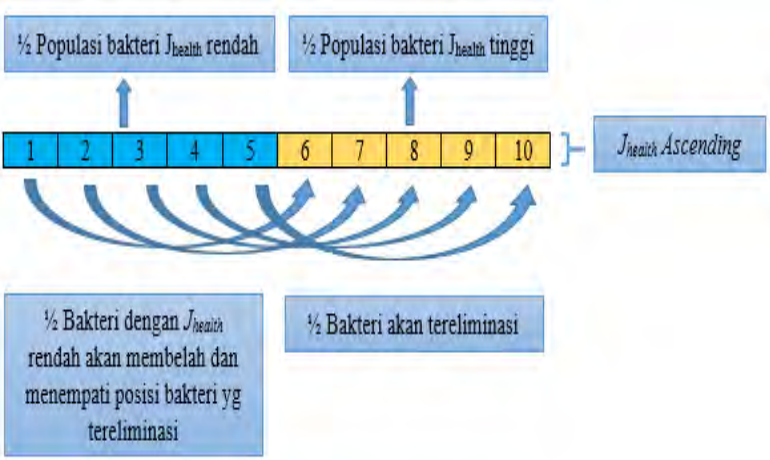
dengan $C(i)$ yang merupakan ukuran dari langkah yang diambil dalam arah yang random yang dispesifikasikan dengan *tumble*.

3.3.2.3 Tahapan Reproduksi

Pada tahap ini, semakin tidak sehat bakteri maka pada akhirnya akan mati dan tiap-tiap bakteri yang sehat (bakteri yang menghasilkan nilai fungsi objektif yang minimal) secara aseksual terpisah menjadi dua bakteri yang kemudian berada di lokasi yang sama. Proses ini membuat jumlah bakteri dalam populasi konstan.

Dalam tahap reproduksi ini, nantinya setengah dari populasi akan dieliminasi dan setengahnya lagi akan membelah diri untuk menggantikan bakteri yang tereliminasi. Eliminasi dari bakteri ini berdasarkan nilai dari J_{health} dari bakteri yang di *sorting* secara *ascending*. Untuk lebih jelasnya dapat dilihat dari Gambar 3.3.

Misalkan terdapat 10 bakteri dalam populasi. Nilai J_{health} di *sorting* secara *ascending*. Sehingga kotak dengan warna biru merupakan setengah dari bakteri dengan nilai J_{health} yang rendah dan kotak dengan warna kuning merupakan bakteri dengan nilai J_{health} yang tinggi sehingga harus dieliminasi. Kemudian bakteri dengan kotak warna biru yang tersisa akan membelah diri menjadi dua dan ditempatkan pada posisi seperti *parent* yang telah dieliminasi sesuai dengan anak panah di atas. Proses ini akan menjaga jumlah bakteri dalam populasi tetap konstan.



Gambar 3.3 Proses Reproduksi Bakteri

Dalam komputasinya, proses reproduksi dilakukan dengan melakukan *splitting*(pembagian populasi bakteri menjadi dua) dengan berdasarkan nilai J_{health}. Nilai J_{health} didapatkan berdasarkan Persamaan 3.5 berikut.

$$J^{i}_{health} = \sum_{j=1}^{N_c+1} J(i,j,k,l) \tag{3.5}$$

3.3.2.4 Tahapan Eliminasi Dispersal

Eliminasi dispersal merupakan kondisi dimana terjadi perubahan dalam lingkungan hidup bakteri. Hal ini bisa disebabkan oleh berbagai macam faktor, seperti: perubahan suhu yang signifikan. Nantinya bakteri yang memiliki nilai J_{health} yang tinggi akan dieliminasi dan disebarkan untuk kembali ke kondisi awal secara random dengan memberikan nilai probabilitas yang kecil P_{ed}.

BAB IV
PERANCANGAN PERANGKAT LUNAK

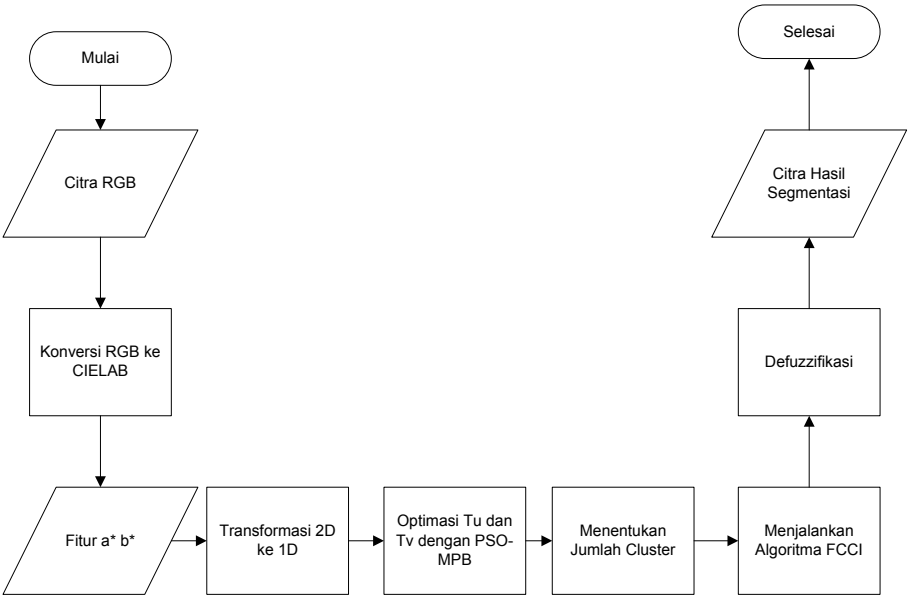
Setelah dijelaskan mengenai metode yang digunakan untuk melakukan segmentasi citra berwarna dengan menggunakan FCCI dan optimasi untuk parameter T_u dan T_v, maka pada Bab 4 akan dijelaskan mengenai perancangan dan desain dari sistem yang digunakan dalam tugas akhir ini.

Perancangan dan desain sistem yang akan dibahas dalam bab ini meliputi desain secara umum dari sistem, perancangan data, data-data yang digunakan, diagram alir (*flowchart*) dan *pseudocode*.

4.1 Desain Secara Umum

Secara umum, segmentasi citra dengan menggunakan algoritma FCCI ini tidak jauh berbeda dengan proses-proses segmentasi menggunakan algoritma lain. Dalam sistem ini terdapat beberapa tahapan. Tahap pertama adalah, konversi warna dari ruang warna RGB ke dalam ruang warna CIELAB. Proses ini bertujuan untuk mendapatkan dua fitur warna yang akan digunakan yaitu a dan b (K=a*,b*).Tahap kedua adalah, melakukan transformasi dari 2D ke 1D dengan *columnwise*. Hal ini dilakukan karena komputasi akan menjadi lebih mudah jika berada dalam 1D.

Kemudian tahap ketiga yaitu, mendapatkan jumlah *cluster* yang tepat untuk setiap citra yang diproses. Langkah keempat yang dilakukan adalah menjalankan algoritma FCCI sesuai dengan jumlah *cluster* yang sesuai. Kemudian langkah terakhir adalah melakukan defuzzifikasi *cluster*. Diagram alir untuk keseluruhan proses segmentasi dapat dilihat pada Gambar 4.1.



Gambar 4.1 Diagram Alir Model Sistem Secara Umum

4.2 Perancangan Data

Perancangan data merupakan hal penting untuk diperhatikan karena diperlukan data yang tepat agar perangkat lunak beroperasi secara benar. Data yang diperlukan dalam pengoperasian perangkat lunak, yaitu data masukan (*input*) yang didapatkan dari pengguna, data proses yang dibutuhkan dan dihasilkan selama proses eksekusi perangkat lunak, dan data keluaran (*output*) yang memberikan hasil proses pengoperasian perangkat lunak untuk pengguna yang menggunakannya.

4.2.1 Data Masukan *(Input)*

Data masukan (*input*) merupakan data yang dimasukkan oleh pengguna perangkat lunak sebagai *trigger* pada sistem segmentasi citra dengan *Fuzzy Co-Clustering For Images* ini. Data ini berupa citra berwarna yang statis yang diunduh dari database *Berkeley* [BSDS300]. Adapun contoh citra yang digunakan sebagai data masukan dapat dilihat pada Gambar 4.2.

Gambar 4.2 Citra Gambar Berwarna Sebagai *Input* pada Sistem

4.2.2 Data Proses

Data proses merupakan data yang dihasilkan pada proses komputasi dan digunakan pada proses selanjutnya untuk menyelesaikan proses segmentasi citra. Pada sub bab ini dijelaskan bahwa data proses yang digunakan dalam perangkat lunak meliputi nama data, tipe data, dan keterangannya. Kolom kedua pada tabel menunjukkan nama data yang diproses, kolom ketiga berisi tipe dari data tersebut, dan kolom ketiga berisi keterangan. Data proses tersebut ditunjukkan pada tabel 4.1 sampai dengan Tabel 4.3.

Tabel 4.1 Data Proses yang Digunakan Pada Sistem (Bagian Pertama)

No.	Nama Data	Tipe Data	Keterangan
1.	U_{ci}	*double*	Nilai fungsi keanggotan objek yang berada diantara 0 dan 1 yang digunakan untuk mencari jumlah *cluster* optimal.
2.	V_{cj}	*double*	Nilai fungsi keanggotaan fitur yang juga digunakan untuk mencari besarnya u_{ci}.
3.	P_{cj}	*double*	Besarnya nilai pusat cluster dari fitur ke-j terhadap *centroid cluster* ke-c. P_{cj} ini digunakan untuk mencari jarak *Euclidean* D_{cij}.

Tabel 4.2 Data proses yang Digunakan Pada Sistem (Bagian 2)

No.	Nama Data	Tipe Data	Keterangan
4.	D_{cij}	*double*	Parameter D_{cij} merupakan jarak *Euclidean* antara titik data ke-i terhadap *cluster c* dan fitur *j*. D_{cij} digunakan untuk mencari nilai v_{cj} dan u_{ci}.
5.	T_u	*integer*	T_u merupakan parameter bobot yang ada di dalam fungsi objektif FCCI yang berkontribusi dalam menentukan nilai u_{ci}.
6.	T_v	*integer*	T_v merupakan parameter bobot yang ada di dalam fungsi objektif FCCI yang berkontribusi dalam menentukan nilai v_{cj}.

Tabel 4.3 Data Proses yang Digunakan Pada Sistem (Bagian 3)

No.	Nama Data	Tipe Data	Keterangan
7.	C	*integer*	Parameter *C* merupakan parameter yang merepresentasikan jumlah *cluster* yang digunakan untuk melakukan segmentasi terhadap citra *input*.
8.	S_old	*integer*	Nilai validitas *cluster Xie and Beni's* awal yang ditentukan melalui inisialisasi. Parameter ini menunjukkan kualitas cluster yang akan dibandingkan dengan nilai validitas *cluster* yang baru.
9.	S_new	*double*	Nilai validitas cluster *Xie and Beni's* yang baru

4.2.3 Data Keluaran (*Output*)

Ada beberapa data yang dikeluarkan dalam sistem ini, diantaranya adalah parameter yang optimal untuk T_u dan T_v, jumlah *cluster* yang optimal untuk citra yang diproses, nilai validitas *cluster* yang baru dan juga citra hasil segmentasi. Berikut ini citra hasil segmentasi yang dihasilkan oleh sistem ditunjukkan pada Gambar 4.3.

Gambar 4.3 Hasil Keluaran Segmentasi Citra dengan FCCI

4.3 Representasi Algoritma Perangkat Lunak

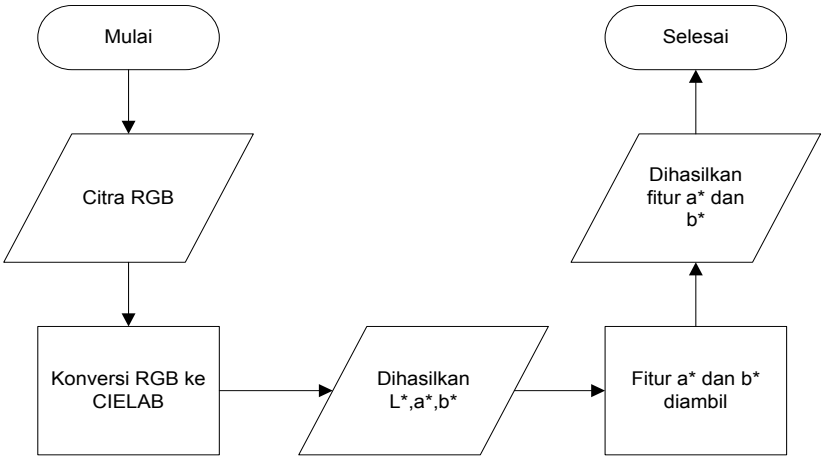
Pada bagian ini akan dijelaskan mengenai algoritma yang digunakan dalam membangun sistem melalui diagram alir (*flowchart*) dan *pseudocode*. Dengan menggunakan diagram alir, dapat diketahui secara jelas dinamisasi data yang diolah dan dihasilkan oleh suatu proses mulai awal hingga akhir proses. Langkah-langkah algoritma sistem dapat dijelaskan dalam beberapa sub bab berikut ini.

4.3.1 Proses Segmentasi Citra dengan FCCI

Dalam melakukan proses segmentasi citra dengan menggunakan algoritma FCCI, ada beberapa tahapan yang dilakukan diantaranya adalah:

4.3.1.1 Proses Konversi Ruang Warna RGB ke CIELAB

Proses konversi citra yang berasal dari ruang warna RGB ke dalam ruang warna CIELAB ini bisa disebut juga sebagai langkah *pre-processing*. Dalam proses ini akan didapatkan tiga fitur yaitu fitur *Luminance* (L*), fitur a*, dan fitur b*. Namun dalam FCCI, hanya fitur a* dan b* saja yang akan digunakan. Hal ini bertujuan untuk membuktikan bahwa pencahayaan (*luminance*) tidak memberikan efek yang signifikan terhadap hasil segmentasi citra. Untuk lebih jelas, proses konversi ini dapat dilihat melalui diagram alir yang ada pada Gambar 4.4 dan *pseudocode* pada Gambar 4.5 berikut ini.



Gambar 4.4 Diagram Alir Konversi Warna dari RGB ke CIELAB

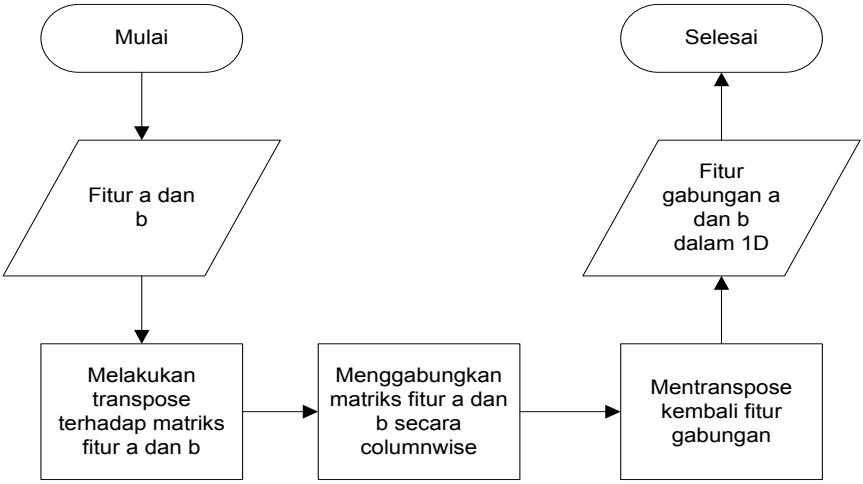
Masukan	Matriks Citra RGB
Keluaran	Fitur (a,b)

```
1.image=imread('295087')
2.labTransformation=makecform('srgb2lab')
3.lab=applycform(image,labTransformation)
4.Menentukan besarnya citra N=mxn
5.Mengambil fitur a dan b
6.Fitur a dan b
7.K=size(ab,3)
```

Gambar 4.5 *Pseudocode* konversi warna dari RGB ke CIELAB

4.3.1.2 Transformasi dari 2D ke 1D

Proses ini dilakukan untuk men-*generate* titik data x_{ij} ke dalam dimensi j , dimana j merupakan fitur $j=1,2$ untuk setiap piksel $i=1,...,N,$ dimana N merupakan jumlah data keseluruhan. Langkah ini menjadi penting karena komputasi akan menjadi lebih mudah jika data dalam 1D daripada dalam 2D. Dalam proses sebelumnya dihasilkan dua fitur a dan b yang berada pada matriks yang berbeda. Kemudian dalam transformasi ini, kedua fitur tersebut ditransformasikan ke dalam dua kolom dimana kedua kolom tersebut dimiliki oleh satu baris piksel. Jadi, kegunaan transformasi ini adalah untuk memudahkan dalam merepresentasikan sebuah piksel dalam citra. Untuk lebih memperjelas proses transformasi dari 2D ke dalam 1D, maka disini diberikan diagram alir dan *pseudocode* yang dapat dilihat pada Gambar 4.6 dan 4.7 berikut.



Gambar 4.6 Diagram Alir Transformasi 2D ke 1D

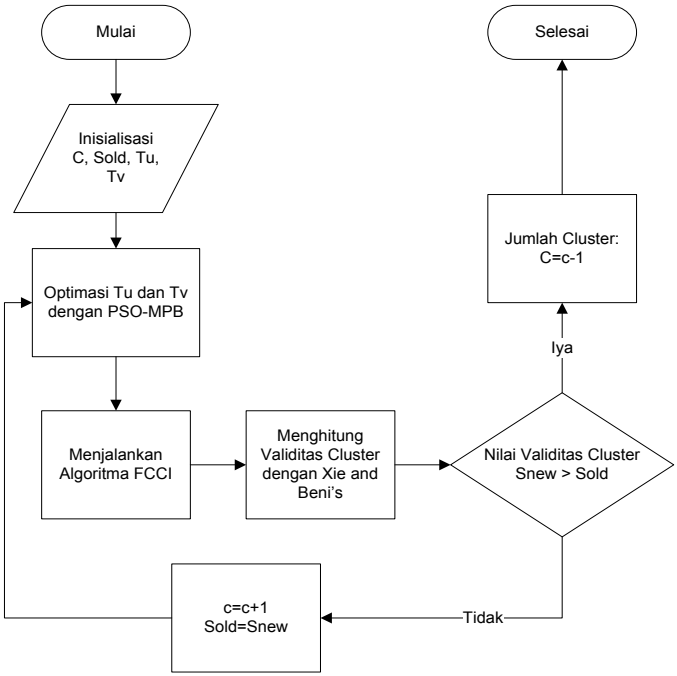
Masukan	Matriks fitur a,b
Keluaran	Xij

```
1. Transpose matrik ab(:,:,1)dan(:,:,2)
2. D1 = reshape(ab(:,:,1)',1,N
3. D2 = reshape(ab(:,:,2)',1,N)
4. Menggabungkan matriks secara
columnwise
5. Xji = [D1;D2]
6.Mentranspose kembali matriks gabungan
4. Xij = Xji'
```

Gambar 4.7 *Pseudocode* Transformasi 2D ke 1D

4.3.1.3 Menentukan Jumlah *Cluster* C

Di dalam proses penentuan jumlah *cluster* ini lah algoritma FCCI dan optimasi parameter T_u dan T_v dengan PSO-MPB dijalankan. Proses ini bertujuan untuk mendapatkan jumlah *cluster* yang tepat untuk citra yang sedang diproses. Dengan jumlah *cluster* yang tepat maka diharapkan hasil segmentasi akan lebih optimal. Untuk lebih memperjelas proses, maka dibuat diagram alir dan *pseudocode* yang ditunjukkan pada Gambar 4.8 dan Gambar 4.9 berikut ini.



Gambar 4.8 Diagram Alir Penentuan Jumlah *Cluster*

Masukan	C,S_old,Tu,Tv,Uci
Keluaran	C(jumlah cluster)

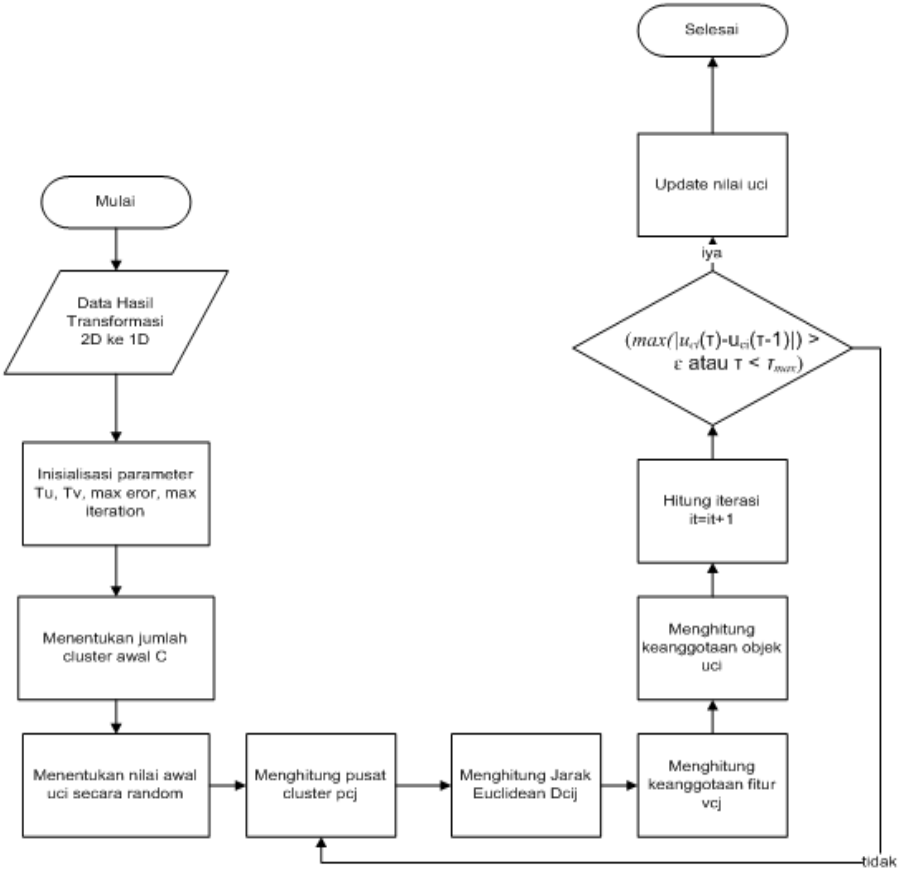
```
1.t=1
2.while(1)
3.  if(t>1)
4.    [Tu,Tv]=BacterialForagingAlgorithm
5.    (Uci,Vcj,Dcij)
6.  end if
7.  t=t+1
8.  [Uci,Vcj,Pcj,Dcij]=
9.  FCCI(C,N,K,Tu,Tv,Xij)
10. S_new=clusterValidity(Uci,Xij,Pcj)
11. if(S_new > S_old)
12.    C=C-1
13.    break
14. else
15.    C=C+1
16.    S_old=S_new
17. end if
18.end while
```

Gambar 4.9 *Pseudocode* Penentuan Jumlah *Cluster*

4.3.1.4 Menjalankan Algoritma FCCI

Langkah yang dilakukan selanjutnya setelah mendapatkan jumlah *cluster* yang tepat untuk citra yang sedang diproses adalah menjalankan kembali algoritma FCCI. Hal ini dilakukan untuk mendapatkan nilai dari fungsi

keanggotaan objek u_{ci} yang tepat, sebelum nanti akhirnya citra didefuzzifikasi. Untuk memperjelas proses yang dilakukan dalam algoritma FCCI, berikut ini diberikan diagram alir dan *pseudocode* yang ditunjukkan pada Gambar 4.10, Gambar 4.11 dan Gambar 4.12.



Gambar 4.10 Diagram Alir Algoritma FCCI

Masukan	maxError,Tu,Tv, maxIteration,Uci_bef
Keluaran	Uci

```
1. it=1
2. for i=1 to jumlah titik data
3.      for j=1 to jumlah fitur
4.          Uci(i,j)=rand()
5.      end for
6. end for
7. Uci_bef=Uci
8. while(true)
9.       sumUci=sum(Uci,2)
10.      Pcj=Uci*Xij
11.       for j=1 to jumlah fitur K
12.           Pcj(:,j)=Pcj(:,j)./sumUci
13.       end for
14.       for i=1 to N titik data
15.         for c=1 to C jumlah cluster
16.             Dcij(c,i,:)=(Xij(i,:)- Pcj
17.             (c,:)).^2
18.         end for
19.       end for
20.       vcj2=zeros(C,K)
21.       for j=1 to jumlah fitur K
22.           vcj1=Uci.*Dcij(:,:,j)/Tv
23.           vcj1=sum(vcj1,2)
24.           vcj2(:,j)=vcj1
25.       end for
```

Gambar 4.11 *Pseudocode* Algoritma FCCI (Bagian 1)

Masukan	maxError,Tu,Tv, maxIteration,Uci_bef
Keluaran	Uci

```
26.            vcj2=exp(-1*vcj2)
27.            vcj3=sum(vcj2,2)
28.        for j=1 to jumlah fitur K
28.            Vcj(:,j)=vcj(:,j)./vcj3
29.        end for
30.            vd1=zeros(K,N)
31.            vd3=zeros(C,N)
32.        for c=1 to jumlah cluster C
33.            for j=1 to jumlah fitur K
34.                vd1(j,:)=Vcj(c,j).*
35.                Dcij(c,:,j)/Tu
36.            end for
37.                vd3=exp(-1*vd3)
38.                vd4=sum(vd3)
39.            for c=1 to jml cluster C
40.                Uci=(c,:)=vd3(c,:)./vd4
41.            end for
42.        it=it+1
43.        if (max(max(abs(Uci-Uci_bef)))<=
44.          maxError||it==maxIteration)
45.        break
46.        end if
47.        Uci_bef=Uci
48. end while
```

Gambar 4.12 *Pseudocode* Algoritma FCCI (Bagian 2)

Diagram alir pada Gambar 4.10 dan *pseudocode* pada Gambar 4.11 sampai Gambar 4.12 menunjukkan rangkaian alur kerja dari algoritma FCCI. *Pseudocode* pada baris 9 sampai dengan baris 13 menunjukkan proses pencarian pusat *cluster c* terhadap fitur *j* (p_{cj}). Dimana dalam proses pencarian pusat *cluster* ini akan mengimplementasikan Persamaan 2.10 yang telah dijelaskan sebelumnya pada Bab 2. Untuk *pseudocode* baris 14 sampai dengan baris 19 ini merupakan *pseudocode* yang digunakan untuk mencari jarak *Euclidean* antara *cluster c*, titik data *i*, dan fitur *j*. Dalam pencarian D_{cij} ini menggunakan Persamaan 2.3.

Setelah mencari jarak *Euclidean*, langkah selanjutnya adalah mencari nilai dari fungsi keanggotaan fitur v_{cj} , dimana dalam *pseudocode* tersebut ditunjukkan pada baris 20 sampai dengan baris 29. Pencarian v_{cj} ini mengimplementasikan Persamaan 2.9. Langkah terakhir adalah mencari nilai dari fungsi keanggotaan objek u_{ci}. Fungsi keanggotaan objek ini menunjukkan hubungan antara tiap *cluster c* terhadap titik data *i*. Untuk implementasi dalam *pseudocode* ini ditunjukkan pada baris 30 sampai dengan baris 41. Pencarian u_{ci} ini menerapkan Persamaan 2.8.

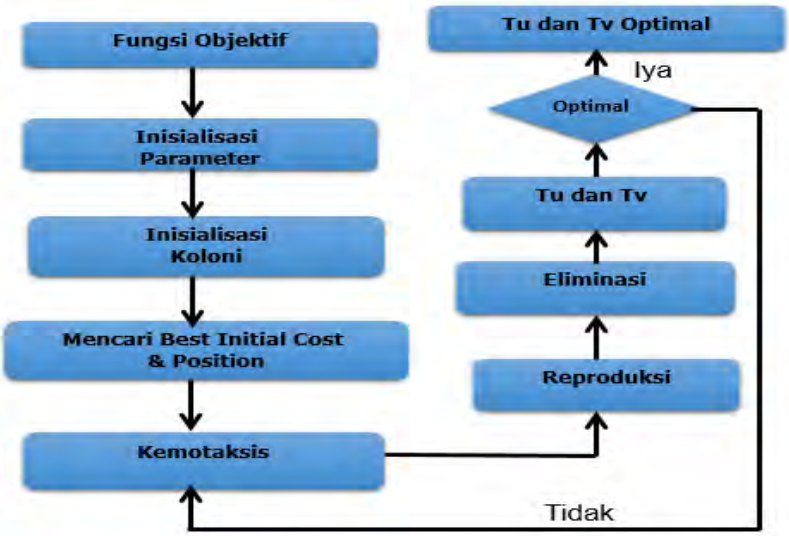
4.3.1.5 Proses Defuzzifikasi

Setelah semua proses dilakukan, maka proses terakhir adalah melakukan defuzzifikasi. Proses ini membutuhkan nilai u_{ci} sebagai nilai fungsi keanggotaan objek yang dihasilkan dari proses sebelumnya. Dalam proses ini, nilai u_{ci} yang dihasilkan akan dipilih yang paling maksimal. Setelah itu, semua titik data akan di *cluster* sesuai dengan nilai u_{ci} yang dimiliki. Proses ini dilakukan sesuai dengan banyaknya titik data dalam citra yang diproses.

4.3.2 Proses Optimasi Parameter T_u dan T_v dengan PSO-MPB

Proses optimasi terhadap parameter T_u dan T_v ini dilakukan untuk meningkatkan hasil dari segmentasi citra yang dilakukan. Kedua parameter ini berada di dalam fungsi objektif algoritma FCCI, dimana nilai T_u memberikan pengaruh terhadap fungsi keanggotaan objek u_{ci} dan nilai T_v memberikan pengaruh terhadap nilai keanggotaan fitur v_{cj}. Diagram alir proses optimasi ini diberikan pada Gambar 4.13.

Proses optimasi ini berlangsung didalam proses pencarian jumlah *cluster* dan dilakukan sebelum menjalankan algoritma FCCI. Jika nilai validitas *cluster S* yang didapatkan belum sesuai dengan ketetapan, maka optimasi kedua parameter dan menjalankan algoritma FCCI ini akan dilakukan kembali, hingga kondisi yang diinginkan tercapai.



Gambar 4.13 Diagram Alir Proses Optimasi Parameter T_u dan T_v

Dari gambaran umum tahapan optimasi di atas, selanjutnya akan dijelaskan lebih detail. Penjelasan lebih lanjut dari proses ini akan diberikan dalam bentuk *pseudocode*. Penjelasan dengan *pseudocode* ini diharapkan dapat memperjelas jalannya proses optimasi.

Untuk memperjelas variabel dan fungsi yang digunakan dalam program, maka diberikan Tabel 4.4 sampai Tabel 4.6 yang berisi daftar variabel dan Tabel 4.7 sampai Tabel 4.8 yang berisi daftar fungsi yang digunakan dalam program.

Tabel 4.4 Daftar Variabel yang Digunakan Pada Proses Optimasi Parameter T_u dan T_v (Bagian 1)

No.	Nama Variabel	Keterangan
1.	`p`	Dimensi dari *search space*
2.	`S`	Jumlah bakteri dalam populasi
3.	`Nc`	Jumlah tahapan kemotaksis yang digunakan
4.	`Ns`	Batasan langkah yang dilakukan dalam tahapan kemotaksis bagian *swim*
5.	`Nre`	Jumlah tahapan reproduksi yang dilakukan
6.	`Ned`	Jumlah tahapan eliminasi dispersal yang dilakukan
7.	`Ped`	Probabilitas tiap bakteri akan dieliminasi atau disebarkan
8.	`Ci`	Besarnya pergerakan dari tiap bakteri

Tabel 4.5 Daftar Variabel yang Digunakan Pada Proses Optimasi Parameter T_u dan T_v (Bagian 2)

No.	Nama Variabel	Keterangan
9.	`minValue`	Batas bawah posisi dari kandidat solusi
10.	`maxValue`	Batas atas posisi dari kandidat solusi
11.	`bacteria`	Kandidat solusi yang dinyatakan sebagai bakteri yang disimpan dalam solusi
12.	`bacteria(i).position`	Variabel yang menyatakan posisi dari bakteri ke-i dalam *search space*
13.	`bacteria(i).health`	Variabel yang menyatakan nilai kesehatan dari bakteri ke-i
14.	`bacteria(i).cost`	Variabel yang menyatakan *cost* dari bakteri ke-i pada *objective function* yang dioptimasi
15.	`bacteria(i).prevCost`	Variabel yang menampung nilai *cost* sebelumnya dari *objective function*

Tabel 4.6 Daftar Variabel yang Digunakan Pada Proses Optimasi Parameter T_u dan T_v (Bagian 3)

No.	Nama Variabel	Keterangan
16.	`optCost`	Variabel yang menampung nilai *cost* dari *objective function* yang paling optimal
17.	`idxOptCost`	Indeks untuk *cost* yang paling optimal
18.	`l`	Indeks untuk *looping* pada eliminasi dispersal
19.	`K`	Indeks untuk *looping* pada reproduksi
20.	`j`	Indeks *looping* untuk tahapan kemotaksis
21.	`i`	Indeks untuk *looping* bakteri
22.	`delta_i`	Variabel yang menampung nilai *delta*(i) untuk menentukan posisi bakteri
23.	`m`	*Looping* untuk *swim length*
24.	`optPosition`	Variabel yang menampung posisi bakteri yang paling optimal

Tabel 4.7 Daftar Fungsi yang Digunakan Dalam Proses Optimasi (Bagian Pertama)

No.	Nama Fungsi	Keterangan
1.	`struct`	Fungsi untuk mendapatkan struktur *array* yang diinisialisasikan masih kosong
2.	`rand`	Fungsi untuk melakukan random terhadap matriks mxn dengan distribusi *uniform*
3.	`fcost`	Fungsi yang digunakan untuk menampung fungsi objektif yang akan dioptimasi
4.	`sortBacteria`	Fungsi untuk melakukan *sorting* terhadap nilai *health cost* dari tiap bakteri
5.	`length`	Fungsi yang digunakan untuk membaca panjang array
6.	`figure`	Fungsi untuk membuat objek grafis
7.	`subplot`	Fungsi yang membagi sebuah *figure* ke dalam kotak persegi sesuai dengan jumlah yang diinginkan
8.	`contour`	Untuk menggambar *contour plot* dari sebuah fungsi
9.	`colormap`	Fungsi untuk mendapatkan warna yang digunakan untuk *figure*
10.	`plot`	Untuk membuat plot 2-D dari nilai yang telah di *set*

Tabel 4.8 Daftar Fungsi yang Digunakan Dalam Proses Optimasi (Bagian Kedua)

No.	Nama Fungsi	Keterangan
11.	`squeeze`	Menghilangkan dimensi tunggal dan mengembalikan sebuah array dengan elemen yang sama tetapi dengan semua dimensi tunggal yang dihilangkan
12.	`axis`	Mengatur batasan koordinat sesuai dengan batas bawah dan batas atasnya
13.	`num2str`	Untuk mengkonversi dari angka ke dalam string
14.	`strcat`	Untuk menyatukan sebuah string secara horisontal
15.	`struct2cell`	Mengkonversi *struct* ke *cell array*
16.	`size`	
17.	`fieldnames`	Fungsi yang mengembalikan sebuah cell array dari sebuah string yang mengandung nama dari objek yang disebutkan
18.	`reshape`	Mengembalikan ukuran dari setiap dimensi pada sebuah *array* ke dalam sebuah vektor
19.	`sortrows`	Untuk melakukan *sorting* terhadap baris secara *ascending*

Dalam melakukan optimasi ini terdapat beberapa tahapan yang signifikan diantaranya adalah inisialisasi variabel, inisialisasi koloni, menemukan *cost* dan posisi awal yang terbaik, kemotaksis yang terdiri atas *tumble* dan s*wim*, reproduksi, dan eliminasi dispersal. Dari tahapan-tahapan tersebut akan dibuat dalam bentuk *pseudocode* yang akan lebih memperjelas jalannya algoritma.

4.3.2.1 Insialisasi Variabel

Dalam langkah ini parameter yang diinisialisasi diantaranya adalah:

a) Dimensi dari *search space*
   Dimensi *search space* ini diinisialisasikan dengan `p`. *Search space* ini merupakan banyaknya dimensi atau variabel dalam fungsi yang akan dioptimasi.
b) Populasi bakteri
   Populasi bakteri merepresentasikan solusi dari problem. Dalam implementasi ini banyaknya bakteri diinisialisasikan dengan `S`.
c) Jumlah langkah kemotaksis
   Banyaknya langkah kemotaksis yang akan dilakukan oleh bakteri diinisialisasikan dengan `Nc`. Langkah kemotaksis dari bakteri terbagi atas dua macam yaitu *tumble* dan *swim*.
d) Batasan *length swim*
   Dalam melakukan pergerakan *swim*, langkah yang diambil oleh bakteri dibatasi sesuai dengan jumlah `Ns` yang ditentukan. `Ns` dalam implementasi BFO ini ditentukan sebanyak 4.
e) Jumlah langkah reproduksi
   Reproduksi bakteri ini dilakukan sesuai dengan banyaknya

langkah yang ditentukan. Langkah reproduksi ini nantinya akan menentukan banyaknya generasi bakteri. Disini `Nre` diinisialisasikan dengan `Nre=4`.

f) Jumlah eliminasi dispersal
Banyaknya eliminasi dan penyebaran bakteri yang akan dilakukan. Dalam implementasi ini, eliminasi dispersal diinisialisasikan dengan `Ned`.

g) Probabilitas eliminasi dispersal
Probabilitas eliminasi dispersal ini dimiliki oleh semua bakteri yang akan dibandingkan dengan nilai yang didapatkan dari random. Jika nilai random ini kurang dari `Ped` maka bakteri akan dieliminasi. Untuk nilai `Ped` ini diinisialisasikan dengan 0.25.

h) Ci
Parameter Ci ini menunjukkan besarnya pergerakan dasar yang dilakukan oleh semua bakteri dalam populasi secara increment. Nilai `Ci` ini disini diinisialisasikan dengan 0.1.

i) Batas atas dan batas bawah
Batasan ini diberikan untuk menentukan *range* dari posisi bakteri yang merepresentasikan solusi dari *problem* optimasi. Dalam implementasi ini diinisialisasikan batas atas dan batas bawah secara berurutan dengan `minValue` dan `maxValue`.

4.3.2.2 Inisialisasi Koloni

Pada tahap ini, setiap bakteri diberikan inisialisasi dalam bentuk *struct* kosong untuk menampung bakteri atau himpunan solusi. Selain itu juga diberikan inisialisasi posisi dari fungsi. Dalam Gambar 4.14 ini diberikan *pseudocode* untuk inisialisasi koloni bakteri.

Masukan	S,minValueTu,maxValueTu,p, minValueTv,maxValueTv
Keluaran	bacteria(i).position

```
1. for i=1 to banyaknya bakteri S
2.    bacteria(i).position(1,1)=(maxValueTu-
3.minValueTu)*rand(p,1)+minValueTu
4.bacteria(i).position(2,1)=(maxValueTv-
5.minValueTv)*rand(p,1)+minValueTv
6.end for
```

Gambar 4.14 *Pseudocode* Inisialisasi Koloni

4.3.2.3 Menemukan *Cost* dan Posisi Awal Terbaik

Setelah setiap bakteri memiliki posisi awal masing-masing, maka langkah selanjutnya adalah memberikan inisialisasi awal untuk *health cost* dari bakteri, *cost*, dan *previous cost*. Penentuan posisi awal terbaik ini bertujuan untuk memberikan nilai *cost* dan posisi awal yang baik untuk masing-masing bakteri. Dalam menemukan posisi ini formula yang diimplementasikan adalah Persamaan 3.3.

Setelah proses update dilakukan dengan mengimplementasikan Persamaan 3.3 tersebut, maka langkah selanjutnya adalah membandingkan hasil yang diperoleh dengan nilai *cost* dan posisi yang dianggap optimal sebelumnya. Setelah itu nilai yang lebih kecil akan diambil sebagai nilai cost dan posisi yang paling optimal. Untuk lebih jelasnya tentang implementasi ini dapat dilihat pada Gambar 4.15.

Pada *pseudocode* yang ada pada Gambar 4.15 untuk *update cost* dari *fitness function* yang berada pada baris 3 sampai dengan baris 6 mengimplementasikan Persamaan 3.3 yang berada pada Bab 3.

Masukan	bacteria(i).position,S,objectiveFunction
Keluaran	optPosition,optCost

```
1.for i=1 to banyaknya bakteri S
2.   bacteria(i).health=0.0
3.    bacteria(i).cost=
4.objectiveFunction(bacteria(i).position,
5.sum_uvD, Tu_Uci, Tv_Vcj) +
6.Jcc_cost(bacteria(i).position, bacteria, S)
7.   bacteria(i).prevCost=bacteria(i).cost
8.      if i=1
9.         optCost=bacteria(i).cost
10.         idxOptCost=i
11.      else
12.         if (bacteria(i).cost<optCost)
13.            optCost=bacteria(i).cost
14.         end if
15.      end if
16.end for
17.optPosition = bacteria(idxOptCost).position
```

Gambar 4.15 Mencari *Cost Fitness Function* dan Posisi Terbaik

4.3.2.4 *Pseudocode* Tahapan *Chemotaxis*

Kemotaksis ini merupakan langkah optimasi yang berada di dalam perulangan dari reproduksi dan eliminasi dispersal. Di dalam tahapan kemotaksis ini terdapat dua pergerakan yang dilakukan oleh bakteri, yang pertama adalah *tumble* untuk mencari orientasi arah pergerakan yang akan dilakukan oleh bakteri dan selanjutnya bakteri akan melakukan *swim*. Berikut ini *pseudocode* kemotaksis bagian *tumble* ditunjukkan pada Gambar 4.16.

Masukan	bacteria(i).position,S,Ned,Nre,Nc
Keluaran	optPosition,optCost

```
1.for l=1 to banyaknya eliminasi dispersal Ned
2.   for K=1 to banyaknya reproduksi Nre
3.       for j=1 to banyaknya kemotaksis Nc
4.           for i=1 to banyaknya bakteri S
5.           bacteria(i).health=0.0
6.           end for
7.               for i=1 to S
8.             delta_i(i)=(1-(-1))*rand(p,1)+(-1)
9.             bacteria(i).position=
10.            bacteria(i).position+(Ci*delta_i)/
11.            sqrt(delta_i'*delta_i)
12.            bacteria(i).prevCost=
13.            bacteria(i).cost
14.            bacteria(i).cost= objectiveFunction
15.            (bacteria(i).position, sum_uvD,
16.            Tu_Uci, Tv_Vcj)+ Jcc_cost
17.            (bacteria(i).position, bacteria, S)
18.            bacteria(i).health=
19.             bacteria(i).health+bacteria(i).cost
20.           if  (bacteria(i).cost < optCost)
21.           optCost = bacteria(i).cost
22.           optPosition = bacteria(i).position;
23.           end if
24.               end for
25.      end for
26.   end for
27.end for
```

Gambar 4.16 *Pseudocode* Gerakan *Tumble*

Setelah melakukan tahapan *tumble*, maka langkah selanjutnya dari pergerakan kemotaksis bakteri adalah *swim*. Pergerakan *swim* ini dilakukan oleh bakteri untuk mendapatkan solusi yang paling optimal. Panjangnya jarak *swim* bakteri ini tergantung dari nilai `Ns` yang diinisialisasikan sebelumnya. Berikut ini *pseudocode* untuk pergerakan kemotaksis *swim* ditunjukkan pada Gambar 4.17 dan Gambar 4.18.

Masukan	bacteria(i).position,S,Ned,Nre,Nc,Ns
Keluaran	optPosition,optCost

```
1.for l=1 to banyaknya eliminasi dispersal Ned
2.    for K=1 to banyaknya reproduksi Nre
3.        for j=1 to banyaknya kemotaksis Nc
4.            for i=1 to banyaknya bakteri S
5.            m=0
6.              while m<Ns
7.                    m=m+1
```

Gambar 4.17 *Pseudocode* Gerakan *Swim* (Bagian 1)

Pada *pseudocode* yang terdapat dalam proses *tumble* yang ditunjukkan oleh Gambar 4.16 dan juga proses *swim* yang terdapat pada Gambar 4.17 dan Gambar 4.17 terdapat implementasi dari Persamaan 3.4. Persamaan ini berlaku untuk proses *tumble* maupun *swim*. Dalam *pseudocode* kemotaksis ini, implementasinya ditunjukkan pada baris 9 sampai dengan baris 11 untuk Gambar 4.16, sedangkan pada Gambar 4.18 terdapat pada Baris 9 dan Baris 10.

Masukan	bacteria(i).position,S,Ned,Nre,Nc,Ns
Keluaran	optPosition,optCost

```
8.if (bacteria(i).cost<bacteria(i)prevCost)
9.  bacteria(i).position = bacteria(i).position
+
10.    (Ci*delta_i)/sqrt(delta_i'*delta_i)
11.     bacteria(i).prevCost =bacteria(i).cost
12.     bacteria(i).cost= objectiveFunction
13.    (bacteria(i).position,sum_uvD,
14.     Tu_Uci, Tv_Vcj)
15.
+Jcc_cost(bacteria(i).position,bacteria,S)
16. if(bacteria(i).cost<optCost)
17.     optCost = bacteria(i).cost
18.     optPosition=bacteria(i).position
19.          end if
20.                  else
21.                       m=Ns
22. end if
23.              end while
24.            end for
25.        end for
26.    end for
27.end for
```

Gambar 4.18 *Pseudocode Swim* (Bagian Kedua)

4.3.2.5 *Pseudocode* Tahapan Reproduksi

Setelah melakukan tahapan kemotaksis, maka tahap selanjutnya adalah melakukan reproduksi. Untuk mempermudah untuk melakukan proses pada tahap reproduksi ini, maka dibuat fungsi untuk melakukan *sorting* terhadap nilai J_{health} yang didapatkan. Untuk memberikan penjelasan tentang jalannya proses ini, maka diberikan *pseudocode* yang dapat dilihat pada Gambar 4.19.

Masukan	bacteria(i).position,S,Ned,Nre, bacteria(i).health, bacteria(i).cost, bacteria(i).prevCost
Keluaran	bacteria(i+(S/2)).position, bacteria(i+(S/2)).health, bacteria(i+(S/2)).cost, bacteria(i+(S/2)).prevCost

```
for l=1 to banyaknya eliminasi dispersal Ned
   for K=1 to banyaknya reproduksi Nre
       bacteria = sortBacteria(bacteria,2)
          for i=1 to S/2
          bacteria(i+(S/2)).position =
          bacteria(i).position
          bacteria(i+(S/2)).health =
          bacteria(i).health
          bacteria(i+(S/2)).cost =
          bacteria(i).cost
          bacteria(i+(S/2)).prevCost =
          bacteria(i).prevCost
          end for
   end for
end for
```

Gambar 4.19 *Pseudocode* Tahap Reproduksi 4.3.2.6

Pseudocode Tahapan Eliminasi Dispersal

Tahap eliminasi dispersal merupakan tahap terakhir dalam optimasi parameter T_u dan T_v. Pada tahap ini, bakteri yang tidak memiliki *cost* optimal yang mendekati solusi, maka akan dieliminasi dengan dikembalikan ke kondisi awal dan disebarkan (*dispersed*). Berikut ini *pseudocode* untuk tahap eliminasi dispersal ditunjukkan pada Gambar 4.20.

Masukan	bacteria(i).position,S,Ned,Nre, bacteria(i).health,bacteria(i).cost, bacteria(i).prevCost,Ped
Keluaran	optCost,optPosition

```
for l=1 to banyaknya eliminasi dispersal Ned
   for i=1 to banyaknya bakteri S
   prob=rand()
      if (prob<Ped)
          bacteria(i).position=        (maxValue-
minValue)
          *rand(p,1)+minValue
          bacteria(i).health=0.0
          bacteria(i).cost=fcost(bacteria(i).
          position)
          bacteria(i).prevCost = bacteria(i).cost
          if (bacteria(i).cost <optCost)
              optCost = bacteria(i).cost
              optPosition= bacteria(i).position
          end if
      end if
   end for
end for
```

Gambar 4.20 *Pseudocode* Eliminasi Dispersal

BAB V
IMPLEMENTASI

Setelah pada Bab 4 dijelaskan mengenai proses segmentasi citra dengan menggunakan algoritma FCCI, maka pada Bab 5 ini akan dijelaskan mengenai tahap implementasi yang meliputi algoritma dan kode program yang terdapat dalam perangkat lunak.

5.1 Lingkungan Implementasi

Spesifikasi perangkat keras dan perangkat lunak yang digunakan dalam segmentasi citra dengan algoritma FCCI ini, ditampilkan pada Tabel 5.1.

Tabel 5.1 Lingkungan implementasi perangkat lunak

Perangkat Keras	Prosesor : Intel(R) Core(TM) i3 CPU M350 @ 2.27 GHz Memori : 3 GB
Perangkat Lunak	Sistem Operasi : Microsoft Windows 7 Ultimate 32-bit Perangkat Pengembang : MATLAB 7.6.0

5.2 Implementasi Program Utama

Pada program utama ini terdiri dari beberapa proses, yaitu konversi citra dari ruang warna RGB ke dalam ruang warna CIELAB, transformasi dari 2D ke dalam bentuk 1D, pencarian jumlah *cluster* yang tepat, menjalankan algoritma FCCI, dan defuzzifikasi.

Data *input* pada sistem segmentasi citra dengan algoritma FCCI ini adalah citra berwarna yang statis. Berikut ini merupakan kode program utama yang dapat dilihat pada Kode Sumber 5.1, Kode Sumber 5.2 dan Kode Sumber 5.3.

```
1.  image = imread('295087.jpg');
2.  (2)Konversi RGB to CIELAB
3.  labTransformation = makecform('srgb2lab');
4.  lab = applycform(image, labTransformation);
5.  m = size(image,1);
6.  n = size(image,2);
7.  image_cluster = zeros(m,n);
8.  N = m*n;% Number of data point of image
9.  ab = zeros(m,n);
10. ab(:,:,1)=lab(:,:,2);
11. ab(:,:,2)=lab(:,:,3);
12. K = size(ab,3); % Number of Feature
13. % (3)Transformasi 2D ke 1D
14. D1 = reshape(ab(:,:,1)',1,N);
15. D2 = reshape(ab(:,:,2)',1,N);
16. Xji = [D1;D2]; % Xij = [D1;D2];
17. Xij = Xji';
18. % (4) Finding the number of cluster
19. % (4.1)
20. C =3;
21. S_old = 50;
22. Tu = 10;
23. Tv = 9*10^7;
24. t = 1;
```

Kode Sumber 5.1 Kode Program Utama (Bagian 1)

```
24. while (1)
25.     % (4.2)
26.     % Optimize Tu, Tv by Pso Algorithm
27.     fprintf('*------* Loop %d\n',t);
28.     if(t>1)
29.         fprintf('* Optimizing Tu, Tv using
    Bacterial Foraging Optimization\n');
30.         [Tu, Tv] =
    BacterialForagingAlgorithm(Uci, Vcj, Dcij);
31.         fprintf('* Found Tu = %.f, Tv = %.f
    \n', Tu,Tv);
32.     end
33.     t = t+1;
34. while (1)
35.     % (4.2)
36.     % Optimize Tu, Tv by Pso Algorithm
37.     fprintf('*------* Loop %d\n',t);
38.  % (4.3)
39.     fprintf('* Calculate Uci, Vcj, Pcj,
    and Dcij using FCCI Algorithm\n');
40.  % (4.4)
41.     fprintf('* Calculate Xie and Beni
    cluster validity S_new\n');
42.     S_new = clusterValidity(Uci,Xij,Pcj);
43.     fprintf('* S_new = %f\n',S_new);
44.  if (S_new > S_old)
45.         C = C-1;
46.         break
47.     else
48.         C = C+1;
49.         S_old = S_new;
50.     end
51. end
```

Kode Sumber 5.2 Kode Program Utama (Bagian 2)

```
52.  fprintf('\n');
53.  fprintf('* Optimal Found for Tu = %.f, Tv
     = %.f \n', Tu,Tv);
54.  fprintf('* Optimal Cluster = %d\n',C);
55.  % (5)FCCI algorithm
56.  fprintf('* Calculate Uci, Vcj, Pcj, and
     Dcij using FCCI Algorithm With optimal
     Cluster, Tu, and Tv\n');
57.  [Uci, Vcj, Pcj, Dcij] = FCCI(C, N, K, Tu,
     Tv, Xij);
58.  % (6) Defuzzify
59.  [Uci_final Uci_idx]= max(Uci);
60   s=1;
61.  for l=1:m
62.      image_cluster(l,:) = Uci_idx(s:s+(n-
     1));
63.      s=s+n;
64.  end
65.  % Menampilkan hasil
66.  pixel_labels = reshape(image_cluster,m,n);
67.  ei = imerode(pixel_labels,ones(3));
68.  di = imdilate(pixel_labels,ones(3));
69.  boundaries = ei~=pixel_labels |
     di~=pixel_labels;
70.  output = imoverlay(image, boundaries, [1 1
     1]);
71.  figure, imshow(output)
72.  % Liu and Yang accuration evaluation
73.  [F] = accurationTest(Xij, image_cluster,
     Pcj, C, N)
```

Kode Sumber 5.3 Kode Program Utama (Bagian 3)

Proses pertama yang dilakukan pada Kode Sumber 5.1 adalah membaca data masukan dengan menggunakan fungsi MATLAB yang ditunjukkan pada baris Baris 1. Data masukan

merupakan citra statis yang berukuran 321x481 atau 481x321. Kemudian dilanjutkan dengan proses melakukan konversi citra dari ruang warna RGB ke dalam ruang warna CIELAB yang ditunjukkan pada Baris 3 sampai Baris 12.

Proses selanjutnya merupakan proses melakukan transformasi dari 2D ke 1D yang ditunjukkan pada Baris 14 sampai dengan Baris 17. Kemudian, tahap selanjutnya adalah menemukan jumlah cluster yang sesuai untuk citra yang diproses. Tahap ini ditunjukkan pada Baris 20 sampai dengan Baris 54. Di dalam proses ini pencarian jumlah *cluster* ini, terdapat proses-proses besar di dalamnya yang akan dijelaskan dalam sub bab berikutnya.

Dari proses pencarian jumlah cluster, dihasilkan cluster yang tepat untuk sebuah citra. Setelah ditemukan jumlah *cluster* yang tepat, maka langkah selanjutnya adalah menjalankan algoritma FCCI dimana kode sumbernya ditunjukkan pada Baris 56 dan Baris 57. Setelah proses dari algoritma FCCI dijalankan maka langkah selanjutnya adalah melakukan defuzzifikasi yang kode sumbernya ditunjukkan pada Baris 59 sampai dengan Baris 64. Kemudian tahap terakhir adalah menampilkan kembali citra keluaran sebagai citra yang telah disegmentasi. Kode program untuk proses ini ditunjukkan pada Baris 66 sampai dengan Baris 71. Untuk mengetahui nilai dari evaluasi kuantitatif segmentasi dengan metode *Liu and Yang's* maka kode programnya ditunjukkan pada baris 72 dan baris 73.

5.3 Implementasi Algoritma FCCI

Pada sub bab ini akan dibahas implementasi dari algoritma FCCI dalam melakukan segmentasi citra. Kode program untuk proses ini akan ditunjukkan pada Kode Sumber 5.4 dan Kode Sumber 5.5.

```
1.  function [Uci, Vcj, Pcj, Dcij] = FCCI(C, N, K, Tu, Tv, Xij)
2.  (1) Initialize the parameters max error limit and max number of iteration
3.  maxError = 10^-2;
4.  maxIteration = 200;
5.  Uci = zeros(C,N);
6.  Vcj = zeros(C,K);
7.  Pcj = zeros(C,K);
8.  Dcij = zeros(C,N,K);
9.  (2) Set iteration number
10. it = 1;
11. % (3) Initialize Uci such that 0 <= Uci <= 1
12. for i=1:size(Uci,1)
13.     for j=1:size(Uci,2)
14.         Uci(i,j)=rand();
15.     end
16. end
17. Uci_bef = Uci;
18. REPEAT
19. while (true)
20. (5) Calculate Pcj
21.  sumUci = sum(Uci,2);
22.     Pcj = Uci * Xij;
23.     for j=1:K
24.         Pcj(:,j)= Pcj(:,j) ./ sumUci;
25.     end
26. (6) Calculate Dcij
27.  for i=1:N
28.         for c=1:C
29.             Dcij(c,i,:)=(Xij(i,:)-Pcj(c,:)).^2;
30.         end
31.     end
```

Kode Sumber 5.4 Kode Sumber Algoritma FCCI (Bagian 1)

```
32.  (7) Calculate Vcj
33.  vcj2 = zeros(C,K);
34.      for j=1:K
35.          vcj1 = Uci.*Dcij(:,:,j)/Tv;
36.          vcj1 = sum(vcj1,2);
37.          vcj2(:,j) = vcj1;
38.      end
39.      vcj2 = exp(-1*vcj2);
40.      vcj3 = sum(vcj2,2);
41.      for j=1:K;
42.          Vcj(:,j) = vcj2(:,j)./vcj3;
43.      end
44.  (8) Calculate Uci
45.  vd1 = zeros(K,N);
46.      vd3 = zeros(C,N);
47.      for c=1:C
48.          for j=1:K
49.              vd1(j,:) =
     Vcj(c,j).*Dcij(c,:,j)/Tu;
50.          end
51.          vd2 = sum(vd1);
52.          vd3(c,:) = vd2;
53.      end
54.      vd3 = exp(-1*vd3);
55.      vd4 = sum(vd3);
56.      for c=1:C
57.          Uci(c,:) = vd3(c,:)./vd4;
58.      end
60.   it=it+1;
61.      % (10) UNTIL max
62.      if (max(max(abs(Uci-Uci_bef))) <=
     maxError || it==maxIteration)
63.          break
64.      end
65.      Uci_bef = Uci;
66.  end
```

Kode Sumber 5.5 Kode Sumber Algoritma FCCI (Bagian 2)

Proses implementasi dari algoritma FCCI ini dijadikan dalam sebuah fungsi. Proses pertama yang dilakukan pada fungsi di atas adalah melakukan inisialisasi parameter batas *error* maksimum dan jumlah iterasi yang kode sumbernya dapat dilihat pada Baris 3 dan Baris 4. Kemudian langkah selanjutnya adalah melakukan inisialisasi untuk parameter U_{ci}, V_{cj}, P_{cj}, dan D_{cij} dengan menyediakan matriks kosong. Kode sumber untuk inisialisasi keempat parameter ini dapat dilihat pada Baris 5 sampai dengan Baris 8.

Setelah melakukan inisialisasi beberapa parameter di atas, langkah selanjutnya adalah melakukan inisialisasi untuk nilai awal dari parameter U_{ci}. Inisialisasi ini dilakukan secara random antara nilai 0 sampai dengan 1. Hal ini dilakukan untuk memberikan nilai awal pada U_{ci}, dimana nantinya parameter ini akan terupdate setelah seluruh proses dalam algoritma dijalankan. Kode sumber untuk proses ini ada pada Baris 12 sampai dengan Baris 17.

Di dalam *while,* proses yang selanjutnya dilakukan adalah menghitung nilai pusat *cluster* c terhadap fitur j atau parameter p_{cj}. Dalam implementasi penghitungan nilai pusat cluster ini digunakan Persamaan 2.10. Kode sumber untuk proses ini berada pada Baris 21 sampai dengan Baris 25.

Setelah nilai pusat *cluster* didapatkan, selanjutnya dicari nilai dari *Euclidean distance* dari sebuah titik data *i* terhadap *cluster c* dan fitur *j*. Untuk mencari nilai D_{cij} ini digunakan Persamaan 2.3. Kode sumber untuk proses ini ditunjukkan pada Baris 27 sampai dengan Baris 31.

Proses selanjutnya setelah nilai *Euclidean distance* ditemukan adalah mencari nilai fungsi keanggotaan fitur dari

sebuah *cluster c* terhadap fitur *j*. Nilai v_{cj} ini dicari dengan menggunakan Persamaan 2.9. Kode sumber untuk proses ini ditunjukkan pada Baris 33 sampai dengan Baris 43.

Proses terakhir yang dilakukan adalah mencari nilai dari fungsi keanggotaan objek dari sebuah titik data *i* terhadap sebuah *cluster c*. Nilai dari u_{ci} ini nantinya akan menentukan sebuah titik data atau piksel tersebut masuk ke dalam *cluster* yang mana. Untuk menghitung nilai u_{ci} ini dalam implementasinya digunakan Persamaan 2.8, sedangkan untuk kode sumber dari proses ini dapat dilihat pada Baris 45 sampai dengan Baris 60. Setelah nilai u_{ci} yang baru didapatkan maka nilai u_{ci} yang lama hasil dari inisialisasi tadi akan diperbarui dengan u_{ci} yang dihasilkan dari proses berdasarkan algoritma FCCI.

5.4 Implementasi Optimasi PSO dengan Modifikasi Perilaku Bakteri

Pada sub bab ini akan dibahas implementasi optimasi parameter T_u dan T_v sebagai parameter yang ikut menentukan hasil dari proses segmentasi. Kode program pada proses optimasi ini dapat dilihat pada Kode Sumber 5.6 sampai 5.11.

```
1.  function [Tu Tv] = PSO(Uci, Vcj, Dcij)
2.  p=2; % Dimension of search space
3.  S=50;    % The number of bacteria
4.  Nc=100; % Number of chemotactic steps
5.  Ns=4; % Limits the length of a swim
6.  Nre=2; % The number of reproduction steps
7.  Ned=2; % The number of elimination-
    dispersal event
```

Kode Sumber 5.6 Proses Optimasi T_u dan T_v (Bagian 1)

```
8.  Ped=0.25; % The probabilty that each bacteria will be eliminated/dispersed
9.  Ci=0.1; % Basic bacteria movement increment size (for all bacteria)
10. minValueTu = 1;
11. maxValueTu = 30;
12. minValueTv = 10^6;
13. maxValueTv = 10^8;
14. C = size(Vcj,1); % cluster
15. K = size(Vcj,2); % jml fitur
16. N = size(Uci,2); % jml piksel
17. vd1 = zeros(K,N);
18. u_ci = zeros(K,N);
19. for c=1:C
20.     for j=1:K
21.         vd1(j,:) = Vcj(c,j).*Dcij(c,:,j);
22.     end
23.     u_ci(c,:) = sum(vd1);
24. end
25. sum_uvD = Uci.* u_ci;
26. sum_uvD = sum(sum(sum_uvD));
27. Tu_Uci = Uci.*log(Uci);
28. Tu_Uci = sum(sum(Tu_Uci));
29. Tv_Vcj = Vcj.*log(Vcj);
30. Tv_Vcj = sum(sum(Tv_Vcj));
31. % Initialize bacteria colony
32. bacteria = struct([]);
33. for i=1:S
34.     bacteria(i).position(1,1) = (maxValueTu - minValueTu) * rand(1,1) + minValueTu;
```

Sumber 5.7 Proses Optimasi T_u dan T_v (Bagian 2)

```
35.  bacteria(i).position(2,1) = (maxValueTv
     - minValueTv) * rand(1,1) + minValueTv;
36.  end
37.  for i=1:S
38.      bacteria(i).health = 0.0;
39.  bacteria(i).cost= objectiveFunction
     (bacteria(i).position,sum_uvD, Tu_Uci,
     Tv_Vcj)+Jcc_cost(bacteria(i).position,
     bacteria,S);
40.  bacteria(i).prevCost = bacteria(i).cost;
41.   % Find best initial cost and position
42.      if (i ==1)
43.          optCost = bacteria(i).cost;
44.          idxOptCost = i;
45.      else
46.          if (bacteria(i).cost < optCost)
47.              optCost = bacteria(i).cost;
48.              idxOptCost = i;
49.          end
50.      end
51.  end
52.  optPosition =
     bacteria(idxOptCost).position;
53.  t = 1; % timer
54.  %Elimination and dispersal loop
55.  for l=1:Ned
56.      % Reprodution loop
57.      fprintf('*--- Elimination-Dispersal
     step = %d \n', l);
58.      for K=1:Nre
59.          % Swim/tumble(chemotaxis)loop
60.          fprintf('*---- Reproduction step
     = %d \n', K);
```

Kode Sumber 5.8 Proses Optimasi T_u dan T_v (Bagian 3)

```
61.  for j=1:Nc
62.  fprintf('*----- Chemotaxis step = %d
     \n', j);
63.                for i=1:S
64.  bacteria(i).health = 0.0;
65.                end
66.                for i=1:S
67.  % Tumble
68.  delta_i = (1-(-1)) * rand(p,1) + (-1);
     % (hi - lo) * r + lo => random i [-1,
     +1]
69.  bacteria(i).position =
     bacteria(i).position +
     (Ci*delta_i)/sqrt(delta_i' * delta_i);
70.  % Update costs of new position
71.  bacteria(i).prevCost =
     bacteria(i).cost;
72.  bacteria(i).cost= objectiveFunction
     (bacteria(i).position,sum_uvD, Tu_Uci,
     Tv_Vcj)+Jcc_cost(bacteria(i).position,
     bacteria,S);
73.  bacteria(i).health =
     bacteria(i).health + bacteria(i).cost;
74.  if (bacteria(i).cost < optCost)
75.  optCost = bacteria(i).cost;
76.  optPosition = bacteria(i).position;
77.  end
78.  % Swim
79.  m=0; % Initialize counter for swim
     length
80.       while m<Ns
81.                 m=m+1;
82.  if (bacteria(i).cost
     <bacteria(i).prevCost)
```

Kode Sumber 5.9 Proses Optimasi T_u dan T_v (Bagian 4)

```
83.  bacteria(i).position =
     bacteria(i).position +
     (Ci*delta_i)/sqrt(delta_i' * delta_i);
84.  bacteria(i).prevCost =
     bacteria(i).cost;
85.  bacteria(i).cost= objectiveFunction
     (bacteria(i).position,sum_uvD, Tu_Uci,
     Tv_Vcj)+Jcc_cost(bacteria(i).position,
     bacteria,S);
86.  if (bacteria(i).cost < optCost)
87.  optCost = bacteria(i).cost;
88.  optPosition = bacteria(i).position;
89.                              end
90.                          else
91.                              m=Ns;
92.                          end
93.                      end
94.                  end % Go to next bacterium
95.  t = t+1;
96.  end
97.  bacteria = sortBacteria(bacteria, 2);
98.  % Split the bacteria (reproduction)
99.          for i=1:S/2
100.             bacteria(i+(S/2)).position
     = bacteria(i).position;
101.             bacteria(i+(S/2)).health =
     bacteria(i).health;
102.             bacteria(i+(S/2)).cost =
     bacteria(i).cost;
103.             bacteria(i+(S/2)).prevCost
     = bacteria(i).prevCost;
104.         end
105.     end %  Go to next reproduction
```

Kode Sumber 5.10 Proses Optimasi T_u dan T_v (Bagian 5)

```
%Eliminatoin and dispersal
    for i=1:S
        prob = rand();
        if (prob < Ped)
            bacteria(i).position(1,1) = (maxValueTu - minValueTu) * rand(1,1) + minValueTu;
            bacteria(i).position(2,1) = (maxValueTv - minValueTv) * rand(1,1) + minValueTv;
            bacteria(i).health = 0.0;
bacteria(i).cost= objectiveFunction (bacteria(i).position,sum_uvD, Tu_Uci, Tv_Vcj)+Jcc_cost(bacteria(i).position, bacteria,S);
bacteria(i).prevCost = bacteria(i).cost;
            if (bacteria(i).cost < optCost)
                optCost = bacteria(i).cost;
                optPosition = bacteria(i).position;
            end
        end
    end
end % Go to next elimination and disperstal
Tu=optPosition(1,1);
Tv=optPosition(2,1);
```

Kode Sumber 5.11 Proses Optimasi T_u dan T_v (Bagian 6)

Proses pertama yang dilakukan pada fungsi di atas adalah inisialisasi parameter yang akan digunakan dalam optimasi. Parameter dalam fungsi objektif yang akan dioptimasi adalah T_u dan T_v, dimana kedua parameter ini akan

memberikan pengaruh dalam mendapatkan nilai fungsi keanggotaan objek nantinya. Untuk implementasi fungsi objektif FCCI ini nanti akan ditunjukkan pada gambar 5.12.

Inisialisasi parameter dalam fungsi ini dapat ditunjukkan pada Baris 2 sampai Baris 9. Dalam parameter tersebut terdapat *dimension of search space* yang diinisialisasikan dengan 2. Hal ini karena dalam problem optimasi ini parameter yang akan dioptimasi adalah 2, yaitu T_u dan T_v. Sedangkan jumlah partikel atau bakteri yang digunakan sebagai himpunan kandidat solusi sebanyak 50, dan seterusnya.

Langkah selanjutnya setelah melakukan inisialisasi parameter adalah memberikan inisialisasi terhadap jangkauan (*range*) dari nilai dua parameter T_u dan T_v. Disini diberikan nilai batas atas dan batas bawah untuk kedua parameter tersebut sesuai dengan ketentuan. Implementasi kode sumber untuk proses ini ditunjukkan pada Baris 10 sampai Baris 13.

Setelah itu, masuk ke dalam proses selanjutnya yaitu memberikan inisialisasi matriks kosong untuk *cluster C*, jumlah fitur *K* dan jumlah piksel *N*. Setelah itu, proses yang dilakukan adalah melakukan proses untuk menentukan nilai keanggotaan objek dan keanggotaan fitur yang nilainya akan dihubungkan dengan kedua parameter T_u dan T_v. Proses ini dapat dilihat pada kode sumber Baris 14 sampai dengan Baris 30.

Memasuki proses optimasi, langkah pertama yang dilakukan adalah melakukan inisialisasi terhadap bakteri dengan memberikan s*truct* kosong. Setelah itu, untuk semua bakteri dari bakteri 1 sampai dengan bakteri ke-S diberikan inisialisasi posisi baik untuk Tu maupun Tv. Proses melakukan inisialisasi posisi bakteri ini dapat dilihat pada kode sumber Baris 32 sampai dengan Baris 36.

Berikutnya, proses yang dilakukan untuk bakteri mulai dari bakteri ke-1 sampai dengan bakteri ke-S adalah menetukan nilai cost berdasarkan fungsi objektif yang akan dioptimasi yaitu fungsi objektif dari *Fuzzy Co-Clustering For Images*. Selain itu nilai dari fungsi `bakteri(i).health` yang menyatakan kesehatan bakteri untuk awalnya di *set* 0. Proses ini dapat dilihat pada kode sumber Baris 38 sampai Baris 40.

Setelah pada proses sebelumnya diberikan inisialisasi *cost* dari fungsi objektif, maka pada langkah berikutnya adalah menentukan kembali nilai posisi dan *cost* yang terbaik atau optimal untuk bakteri mulai dari bakteri ke-1 sampai dengan bakteri ke-S. Jika *cost* dari bakteri ke-i kurang dari optimal *cost*, maka nilai optimal *cost* yang baru diambil yang lebih kecil. Sedangkan untuk posisi yang optimal diambil berdasarkan *cost* yang optimal tersebut. Untuk implementasi proses ini dapat dilihat pada kode sumber Baris 42 sampai Baris 52.

Untuk proses selanjutnya Baris 55 sampai Baris 121 adalah proses mulai dari eliminasi dispersal, reproduksi dan proses kemotaksis. Setelah semua proses tersebut dilakukan maka pada hasil akhirnya adalah posisi yang optimal dari parameter T_u dan T_v selain juga ditemukan cost dari fungsi objektif yang paling optimal. Implementasi fungsi objektif FCCI yang dioptimasi ditunjukkan pada Kode Sumber 5.12.

```
1.  function f=objectiveFunction(x,
    sum_uvD, Tu_Uci, Tv_Vcj)
2.  Tu = x(1,1);
3.  Tv = x(2,1);
4.  f = sum_uvD + Tu * Tu_Uci + Tv *Tv_Vcj;
```

Kode Sumber 5.12 Fungsi Objektif FCCI

5.5 Implementasi *Cluster Validity*

Pada sub bab ini akan dibahas mengenai implementasi dalam melakukan evaluasi *cluster* dengan menggunakan *Xie and Beni's Cluster Validity*. Kode program untuk validitas kualitas *cluster* ini ditunjukkan pada Kode Sumber 5.13.

```
1.  function S =clusterValidity(Uci,Xij,Pcj)
2.  C = size(Pcj,1); % cluster
3.  N = size(Uci,2); % jml piksel
4.  Pcj_plus = Pcj(2:C,:);
5.  Pcj_plus(C,:) = Pcj(1,:);
6.  dmin = sum((Pcj_plus - Pcj).^2,2);
7.  dmin = min(dmin);
8.  Dci = zeros(C,N);
9.  for i=1:N
10.     for c=1:C
11.         Dci(c,i)=(Xij(i,1)-Pcj(c,1))^2;
    % fitur j=1 pertama
12.         Dci(c,i)= Dci(c,i)+(Xij(i,2)-
    Pcj(c,2))^2; % fitur j=2 kedua
13.     end
14. end
15. sigma = Uci.^2 .* Dci;
16. sigma = sum(sigma,2);
17. sigma = max(sigma);
18. S = (sigma/N)/dmin^2;
```

Kode Sumber 5.13 *Cluster Validity*

Pada kode evaluasi kualitas *cluster* pada Gambar 5.13 di atas ditunjukkan bahwa variabel S menunjukkan nilai dari validitas *cluster*. Nilai S ini didapatkan berdasarkan Persamaan 2.11. Nilai S ini nantinya akan mengganti nilai S yang diinisialisasikan pada awal proses.

5.6 Implementasi Evaluasi Kuantitatif Segmentasi Citra

Pada subbab ini akan dibahas mengenai implementasi dalam melakukan evaluasi kuantitatif terhadap hasil segmentasi citra. Kode evaluasi kuantitatif segmentasi citra ditunjukkan pada Kode Sumber 5.14.Pada kode evaluasi kuantitatif segmentasi citra ditunjukkan penghitungan hasil evaluasi berdasarkan fungsi yang diusulkan Liu dan Yang. Implementasi ini berdasarkan pada Persamaan 2.14 yang telah dijelaskan sebelumnya pada Bab 2.

```
1.  function [F] = accurationTest(Xij,
    image_cluster, Pcj, C, N)
2.  ei = zeros(C,1);
3.  Ai = zeros(C,1);
4.  G = C;
5.  for i=1:G
6.      Ai(i) =
    size(find(image_cluster==i),1);
7.      pixels = find(image_cluster==i);
8.      dist = zeros(Ai(i),1);
9.      for l=1:Ai(i)
10. dist(l,1) = sqrt((Pcj(i,1)-
    Xij(pixels(l),1))^2 + (Pcj(i,2)-
    Xij(pixels(l),2))^2);
11. end
12.     ei(i) = sum(dist);
13. end
14. F = (1/(1000*N))*sqrt(G)*
    sum(ei.^2./sqrt(Ai));
```

Kode Sumber 5.14 Kode Evaluasi Kuantitatif Segmentasi Citra

BAB VI
UJI COBA DAN EVALUASI

Pada bab ini diuraikan mengenai rangkaian uji coba dan evaluasi yang telah dilakukan untuk segmentasi citra dengan menggunakan FCCI beserta uji coba untuk optimasi terhadap parameter T_u dan T_v menggunakan PSO-MPB. Pembahasan pada bab ini meliputi lingkungan uji coba, skenario uji coba, hasil uji coba, dan evaluasi. Uji coba yang akan dilakukan dalam tugas akhir ini meliputi dua hal yaitu uji coba untuk proses segmentasi dan uji coba proses optimasi. Uji coba proses optimasi dengan PSO-MPB ini dilakukan untuk membuktikan bahwa algoritma yang digunakan sudah benar dapat digunakan untuk melakukan optimasi terhadap fungsi objektif FCCI.

Uji coba dilakukan dengan menggunakan data masukan berupa citra berwarna statis yang di dalamnya terdapat objek yang akan diamati. Kemudian akan dilakukan proses segmentasi sesuai dengan masing-masing skenario yang telah ditentukan sehingga dapat dihasilkan citra yang tersegmentasi. Untuk uji kebenaran optimasi terhadap parameter T_u dan T_v digunakan data uji coba berupa fungsi persamaan nonlinear untuk mengetahui tingkat efisiensi sistem dalam mencari solusi yang terbaik.

6.1 Lingkungan Uji Coba

Lingkungan uji coba yang digunakan dalam pembuatan tugas akhir ini meliputi perangkat lunak dan perangkat keras yang digunakan untuk melakukan uji coba implementasi metode FCCI. Lingkungan uji coba merupakan komputer tempat uji coba sistem yang spesifikasinya dijelaskan pada Tabel 6.1.

Tabel 6.1 Lingkungan Uji Coba Perangkat Lunak

Perangkat Keras	Prosesor : Intel(R) Core(TM) i3 CPU M350 @ 2.27 GHz Memori : 3 GB
Perangkat Lunak	Sistem Operasi : Microsoft Windows 7 Ultimate 32-bit Perangkat Pengembang : MATLAB 7.6.0

6.2 Data Uji

Data uji ini merupakan data yang digunakan untuk melakukan uji coba dan uji kebenaran terhadap metode segmentasi dan metode optimasi. Data ini meliputi data citra statis untuk proses segmentasi dan data berupa persamaan nonlinear untuk proses optimasi.

6.2.1 Data Uji Optimasi Parameter Tu dan Tv

Data uji coba yang digunakan untuk mengetahui kebenaran metode yang digunakan untuk optimasi disini adalah lima persamaan nonlinear yang ditunjukkan pada Tabel 6.2. Penggunaan persamaan nonlinear ini untuk mengetahui tingkat efisiensi sistem dalam mencari solusi yang paling optimal.

Data masukan yang digunakan pada uji coba ini yaitu antara lain sistem persamaan nonlinear *f(x),* domain solusi [xup, xlow], jumlah variabel yang belum diketahui *p*, dan jumlah partikel *S*.

Tabel 6.2 Data Uji Kebenaran Untuk Metode Optimasi Parameter Tu dan Tv

Skenario	Fungsi *f(x)*	Domain Solusi	p	S
1	Fungsi kuadrat	[0,30]	2	50
2	Fungsi Rastrigin	[-5.2,5.2]	10	1000
3	Fungsi Sinus	[3,13]	10	500
4	Fungsi Powell-Quadratic	[-5,5]	4	1000
5	Fungsi Six-Hump Camelback	[-10,10]	2	1000
6	Fungsi Rosenbrock	[-10,10]	2	1000

6.2.2 Data Uji Coba Segmentasi Citra

Tahap uji coba sistem ini menggunakan citra berwarna statis yang diperoleh dari *database* Berkeley [BSDS300]. Citra uji coba tersebut terdiri dari 1 citra berwarna statis yang digunakan sebagai uji kebenaran dan 6 citra statis yang digunakan sebagai uji kinerja. Citra uji kebenaran dan citra untuk uji kinerja ditunjukkan pada Lampiran Tabel A.1, Tabel A.2, dan Tabel A.3.

6.3 Skenario Uji Kebenaran Metode Optimasi

Dalam skenario uji ini, pengujian yang akan dilakukan adalah uji kebenaran terhadap metode optimasi yang digunakan untuk melakukan optimasi terhadap fungsi objektif FCCI dengan tujuan untuk mendapatkan nilai parameter T_u dan T_v yang optimal. Untuk mengetahui keefektifan metode yang digunakan dalam mencari solusi yang optimal, disini digunakan enam fungsi persamaan nonlinear yang masing-masing akan dijelaskan dalam skenario-skenario berikut ini.

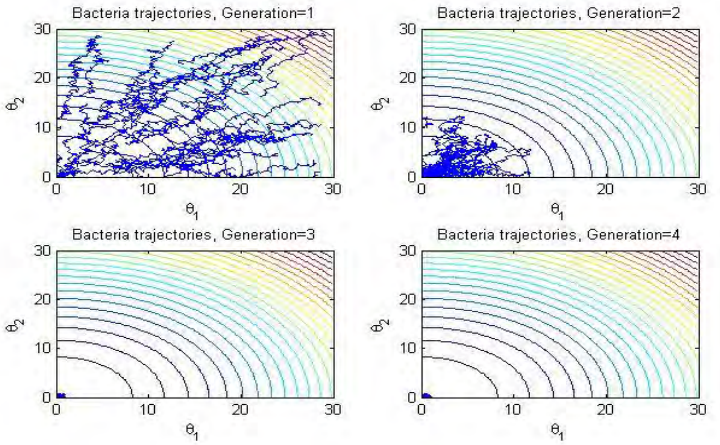
6.3.1 Uji Kebenaran Skenario 1

Skenario uji coba pertama dilakukan dengan menggunakan fungsi kuadrat berikut ini:

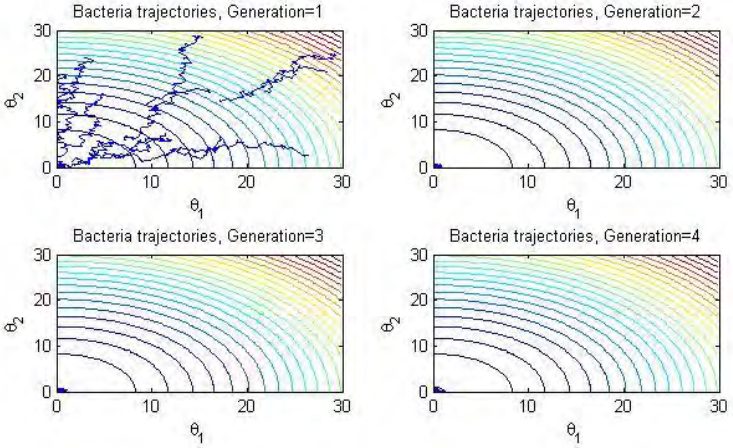
$$f(x)=x_1^2+x_2^2$$

Fungsi ini termasuk fungsi nonlinear yang memiliki solusi minimum yang berada pada titik (0,0) dengan nilai fungsi objektif $f(x)$=0. Untuk mengetahui konvergensi dari metode yang digunakan, berikut ini ditunjukkan *contour plot* 2D pada Gambar.

Dalam *contour plot* yang pertama digambarkan tahap eliminasi dispersal yang pertama. Masing-masing *sub-plot* gambar menggambarkan grafik jejak bakteri tiap tahap reproduksi dari reproduksi ke-1 sampai dengan ke-`Nre`. Hal ini berlaku juga untuk gambar kedua. Dimana dalam gambar kedua menunjukkan eliminasi dispersal yang kedua hingga bakteri mencapai tahap konvergen.



Gambar 6.2 *Contour Plot* 2D Hasil Uji Skenario Optimasi PSO dengan Fungsi Kuadrat (Bagian 1)



Gambar 6.1 *Contour Plot* 2D Hasil Uji Skenario Optimasi PSO dengan Fungsi Kuadrat (Bagian 2)

Berdasarkan Gambar 6.1 dan Gambar 6.2, metode optimasi ini bisa dikatakan konvergen. Dari hasil uji coba, didapatkan nilai *optCost* sebesar 2.0643e-006 dan o*ptPosition* untuk dua variabel yang dicari nilainya berada di 0.0014 dan 0.0001. Kisaran nilai dari *optCost* berada diantara 10^{-7} sampai dengan 10^{-6}, jadi sangat kecil sekali, mendekati angka nol.

6.3.2 Uji Kebenaran Skenario 2

Skenario uji coba kedua dilakukan dengan menggunakan fungsi Rastrigin berikut.

$$f(x) = \sum_{i=1}^{10} (x_i^2 - 10\cos(2\pi x_i) + 10) \quad |x_i| \leq 5.2.$$

Fungsi ini termasuk fungsi non-linear multimodal sehingga sulit diselesaikan karena memiliki *search space* dan jumlah *local minima* yang besar. Solusi minimum fungsi Rastrigin berada pada titik x=(0,0,0,0,0,0,0,0,0,0) dengan nilai fungsi objektif $f(x) = 0$. Data masukan yang digunakan pada uji coba ini antara lain domain solusi [5.2. -5.2], jumlah variabel yang belum diketahui p yaitu 10, dan jumlah bakteri S yang digunakan yaitu 1000.

Berdasarkan hasil uji coba, metode optimasi dengan PSO berkarakteristik perilaku bakteri ini telah konvergen. Hasil uji coba pada Tabel 6.3 menunjukkan metode optimasi ini konvergen dengan nilai fungsi objektif *f(x) optCost*=0.0194, *optPosition*=(0.0006,-0.1008,-0.0284,0.0400,0.0222,-0.0449, 0.0124,-0.0198,-0.0466,-0.0220).

Tabel 6.3 Tabel Hasil Skenario Optimasi dengan Fungsi Rastrigin

	optPosition		optCost	
	PSO-MPB	**Titik Optimal**	**PSO-MPB**	***f(x)***
x1	0.0006	0	0.0194	0
x2	-0.1008	0		
x3	-0.0284	0		
x4	0.0400	0		
x5	0.0222	0		
x6	-0.0449	0		
x7	0.0124	0		
x8	-0.0198	0		
x9	-0.0466	0		
x10	-0.0220	0		

6.3.3 Uji Kebenaran Skenario 3

Uji skenario kedua dilakukan dengan menggunakan fungsi objektif *f(x)* Sinus, sebagai berikut:

$$\max\ f(x) = -\sum_{i=1}^{10} \left[\sin(x_i) + \sin\left(\frac{2x_i}{3}\right) \right]$$

Nilai fungsi objektif f(x) pada fungsi ini adalah -12.1395. Data masukan yang digunakan antara lain, *domain* solusi [3,13], jumlah variabel yang belum diketahui yaitu 10, dan jumlah bakteri yang digunakan yaitu 500.

Berdasarkan hasil uji coba metode optimasi ini dengan fungsi sinus di atas, dapat dilihat bahwa metode dengan PSO

ini konvergen dan fungsi tersebut menemukan nilai *f(x)* yang optimal *optCost* pada angka -12.1395 dan posisi yang optimal *optPosition* untuk kesepuluh variabel yang belum diketahui berada pada *x*=(5.3780, 5.2802, 5.3978, 5.3951, 5.3270, 5.3357, 5.2152, 5.3519, 5.2966, 5.4232).

Tabel 6.4 Hasil Uji Skenario Optimasi dengan Menggunakan Fungsi Objektif Sinus

	optPosition		optCost	
	PSO-MPB	**Titik Optimal**	**PSO-MPB**	***f(x)***
x1	5.3780	5.3621	-12.1395	-12.1395
x2	5.2802	5.3622		
x3	5.3978	5.3630		
x4	5.3951	5.3626		
x5	5.3270	5.3622		
x6	5.3357	5.3622		
x7	5.2152	5.3622		
x8	5.3519	5.3603		
x9	5.2966	5.3678		
x10	5.4232	5.3494		

6.3.4 Uji Kebenaran Skenario 4

Skenario uji coba yang keempat dilakukakan terhadap fungsi Powell Quadratic, sebagai berikut:

$$min\ f(x) = (x^1+10x^2)^2 + 5(x^3-x^4)^2 + (x^2-2x^3)^4 + 10(x^1-x^4)^4$$

Solusi fungsi ini terletak pada titik x=(0,0,0,0) dengan nilai fungsi objektif $f(x)$=0.0. Percobaan dilakukan dengan data masukan antara lain *domain* solusi [-5,5], jumlah variabel atau dimensi yang belum diketahui yaitu 4, dan jumlah bakteri yang digunakan yaitu 1000. Hasil uji coba metode dengan fungsi Powell Quadratic dapat dilihat pada Tabel 6.5 di bawah ini.

Tabel 6.5 Hasil Uji Skenario Optimasi dengan Fungsi Powell Quadratic

	optPosition		**optCost**	
	PSO-MPB	**Titik Optimal**	**PSO-MPB**	***f(x)***
x1	-0.0633	0	1.5×10^{-4}	0
x2	0.0062	0		
x3	-0.0027	0		
x4	-0.0055	0		

Berdasarkan hasil uji coba di atas, metode optimasi dengan PSO-MPB dapat dikatakan konvergen dengan nilai fungsi objektif *f(x)* yang optimal *optCost* sebesar 1.5×10^{-4} yang bisa dikatakan sangat kecil dan mendekati 0. Sedangkan untuk posisi yang optimal dari variabel yang belum diketahui *optPosition* berada pada x=(-0.0633, 0.0062, -0.0027, -0.0055).

6.3.5 Uji Kebenaran Skenario 5

Skenario uji coba yang kelima dilakukan terhadap fungsi Six-Hump Camelback

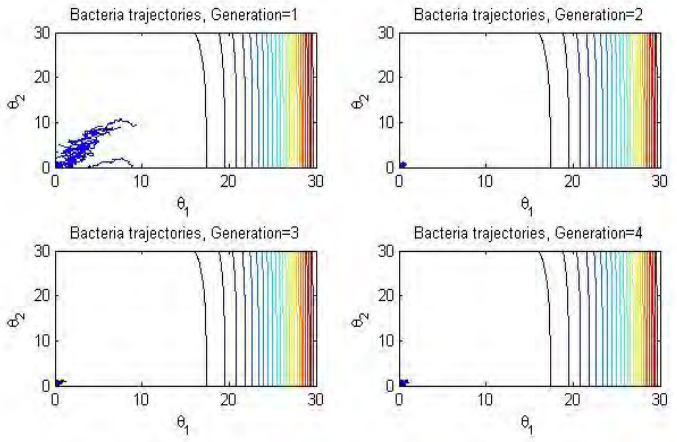
$$\min f(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4 .$$

Fungsi Six-Hump Camelback memiliki dua nilai optimal global, yaitu terletak pada titik x=(-0.08984, 0.71266) atau titik x=(0.08984, -0.71266) dengan nilai fungsi $f(x)$ = -1.0316. Hasil uji coba optimasi dengan fungsi ini dapat dilihat pada Tabel 6.6 berikut.

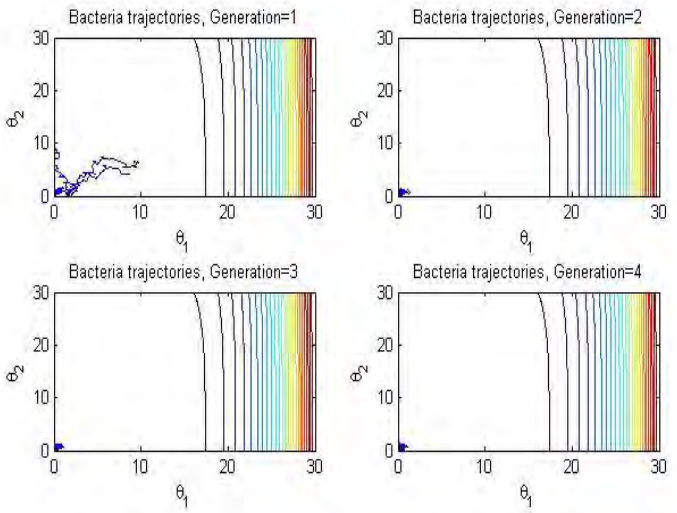
Tabel 6.6 Hasil Uji Skenario Optimasi dengan Fungsi Six-Hump Camelback

	optPosition		**optCost**	
	PSO-MPB	**Titik Optimal**	**PSO-MPB**	***f(x)***
x1	0.0900	(-0.08984,0.71266)	-1.0316	-1.0316
x2	-0.7124	(0.08984,-0.71266)		

Sedangkan untuk hasil percobaan berupa *contour plot* dapat dilihat dalam Gambar 6.3 dan Gambar 6.4 berikut.



Gambar 6.3 *Contour Plot* 2D Hasil Skenario Optimasi dengan Fungsi Six-Hump Camelback (Bagian 1)



Gambar 6.4 *Contour Plot* 2D Hasil Skenario Optimasi dengan Fungsi Six-Hump Camelback (bagian 2)

Percobaan dilakukan dengan memasukkan data masukan, antara lain domain fungsi [-10,10], jumlah variabel atau dimensi *p* yang belum diketahui, yaitu 2, dan jumlah bakteri yang digunakan, yaitu 100.Berdasarkan hasil uji coba di atas, dapat dikatakan bahwa metode optimasi ini telah konvergen. Nilai optimal *optCost* dari fungsi objektif *f(x)* yang didapatkan adalah -1.0316 dan posisi optimal yang didapatkan *optPosition* berada pada x=(0.0900, -0.7124).

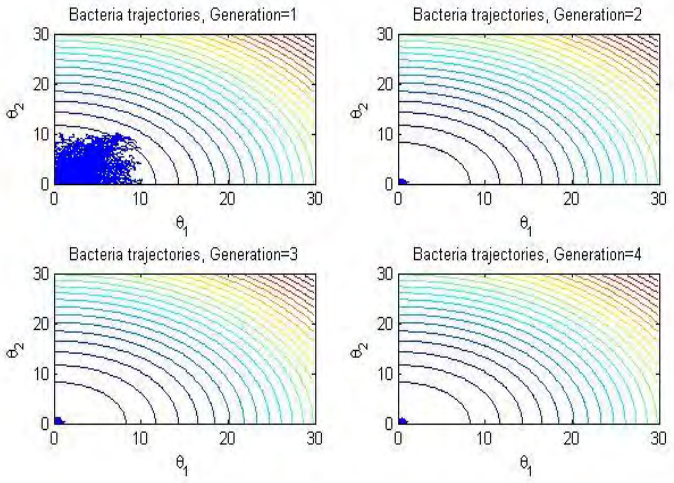
6.3.6 Uji Kebenaran Skenario 6

Pada skenario uji coba keenam ini digunakan fungsi Rosenbrock berikut.

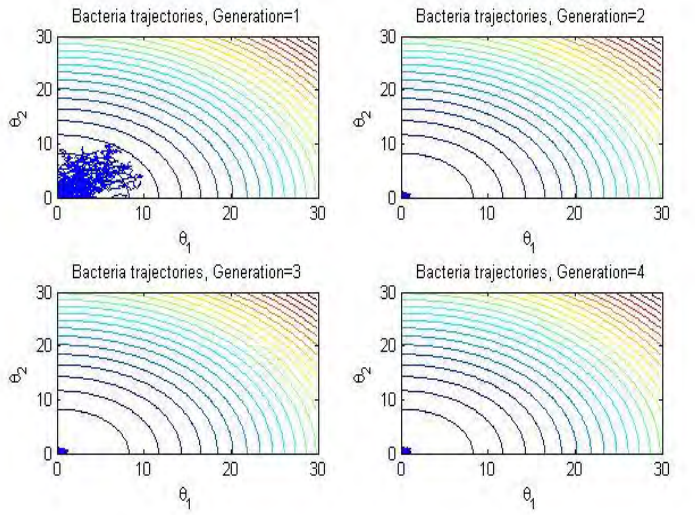
$$f(x) = 100(x_2 - x_1^2) + (1 - x_1)^2 .$$

Solusi minimum fungsi ini terletak pada titik x=(1,1) dengan nilai fungsi $f(x) = 0$. Data masukan yang digunakan pada percobaan ini antara lain, *domain* fungsi [-10,10], jumlah variabel atau dimensi p yang belum diketahui, yaitu 2, dan jumlah bakteri yang digunakan S, yaitu 500.

Gambar 6.5 dan Gambar 6.6 menunjukkan bahwa metode optimasi ini telah konvergen dengan nilai fungsi objektif $f(x)$ yang optimal *optCost* berada pada 1.2767e-005, angka yang kecil sekali mendekati 0. Sedangkan posisi yang optimal *optPosition* berada pada x=(0.9971, 0.9940) yang bisa dikatakan mendekati x=(1,1).



Gambar 6.5 *Contour Plot* 2D Hasil Skenario Optimasi dengan Fungsi Rosenbrock (Bagian 1)



Gambar 6.6 *Contour Plot* 2D Hasil Skenario Optimasi dengan Fungsi Rosenbrock (Bagian 2)

Tabel 6.7 Hasil Uji Skenario Optimasi dengan Fungsi Rosenbrock

	optPosition		**optCost**	
	PSO-MPB	**Titik Optimal**	**PSO-MPB**	***f(x)***
x1	0.9971	1	1.2×10^{-5}	0
x2	0.9940	1		

6.4 Skenario Uji Coba Segmentasi Citra

Pada bagian ini akan dijelaskan mengenai skenario uji coba yang telah dilakukan. Terdapat beberapa skenario uji coba yaitu uji kebenaran metode terhadap citra uji dan uji kinerja segmentasi citra terhadap waktu komputasi. Skenario uji kebenaran adalah sebagai berikut:

1. Uji kebenaran metode segmentasi dengan algoritma FCCI.

Skenario uji kinerja antara lain:

1. Perbandingan hasil segmentasi citra berdasarkan jumlah *cluster* awal yang digunakan pada proses penentuan nilai u_{ci} yaitu, 3,5 dan 7.
2. Perbandingan hasil segmentasi citra berdasarkan nilai parameter T_u awal yang digunakan dalam proses optimasi yang akan berkontribusi dalam penentuan nilai fungsi keanggotaan objek u_{ci} yaitu 10,20, dan 30.
3. Perbandingan hasil segmentasi citra berdasarkan nilai parameter T_v yang digunakan dalam proses optimasi dan akan mempengaruhi nilai dari fungsi keanggotaan fitur v_{cj} yaitu $9x10^6$, $9x10^7$, dan $9x10^8$.
4. Perbandingan hasil segmentasi citra berdasarkan ukuran piksel citra yang berbeda-beda, antara lain, 481x321, 361x241, dan 241x161.
5. Perbandingan hasil segmentasi citra berdasarkan jumlah bakteri yang digunakan sebagai domain solusi dari parameter T_u dan T_v, yaitu 50,100, dan 150.
6. Perbandingan hasil segmentasi citra berdasarkan jumlah langkah kemotaksis N_c yang dilakukan bakteri untuk menentukan parameter yang optimal, yaitu, 50, 100, dan 200.

6.4.1 Uji Kebenaran: Uji Kebenaran Metode *Fuzzy Co-Clustering* terhadap Citra Masukan

Pada skenario ini dilakukan uji kebenaran terhadap metode FCCI dengan menggunakan evaluasi kualitas *cluster* dan evaluasi kuantitatif segmentasi. Hasil uji coba skenario ditunjukkan pada Tabel 6.8.

Tabel 6.8 Tabel Uji Kebenaran Metode FCCI

Kategori	Madasu Hanmandlu	Uji Coba
Jumlah *Cluster*	3	3
S(Validity Cluster)	0,00080	0,000078
F(I)	455,281	449,416

Hasil uji skenario menunjukkan jumlah *cluster* yang diperoleh sama dengan jumlah *cluster* pada Madasu Hanmandlu *et al*. Untuk hasil hasil *cluster validity* (S) juga hampir mendekati hasil dari paper rujukan. Hasil segmentasi pada citra 295087.jpg ditunjukkan pada Gambar 6.7. Untuk hasil uji kebenaran lainnya dapat dilihat pada Lampiran Tabel A.28 sampai Tabel A.31.

(a) **(b)**

Gambar 6.7 Hasil Uji Kebenaran Metode FCCI (a) Citra asal 295087.jpg (b) Citra hasil segmentasi

6.4.2 Uji Kinerja Skenario 1: Perbandingan Hasil Segmentasi Citra Berdasarkan Inisialisasi *Cluster* Awal

Pada skenario ini akan dibandingkan nilai *cluster* awal yang digunakan dalam melakukan segmentasi yaitu dengan nilai *cluster* awal 3,5, dan 7. Uji coba skenario ini dilakukan terhadap enam citra berwarna diantaranya adalah 42049.jpg, 86000.jpg, 109053.jpg, 147091.jpg, 189080.jpg, 296059.jpg dan 216081.jpg. Hasil uji skenario ditunjukkan pada Tabel 6.9.

Pada citra 42049.jpg, 86000.jpg, 109053.jpg, 147091.jpg, 189080.jpg, 296059.jpg menunjukkan bahwa setiap nilai *cluster C* menghasilkan nilai fungsi keanggotaan objek u_{ci}, sehingga mempengaruhi nilai dari validitas *cluster S*. Dengan inisialisasi jumlah *cluster* awal yang berbeda yaitu 3,5, dan 7 dapat dilihat bahwa waktu komputasi yang dibutuhkan akan berbeda antara citra yang menggunakan nilai awal 3 dengan 5 atau lainnya.

Berdasarkan Tabel 6.9 dapat dilihat bahwa semakin besar nilai inisialisasi *cluster* di awal proses, maka waktu komputasi juga akan semakin besar. Kenaikan jumlah *cluster* dari 3 ke 5 memberikan kenaikan waktu komputasi sebesar 8% dan kenaikan dari *cluster* 5 ke *cluster* 7 memberikan kenaikan sebesar 15%. Dalam Tabel 6.9 dapat dilihat bahwa inisialisasi *cluster* pada awal proses juga memberikan pengaruh terhadap nilai dari kuantitatif segmentasi citra. Semakin besar inisialisasi *cluster*, maka nilai kuantitatif segmentasi citra (F) juga semakin meningkat. Jika dilihat dari segi waktu komputasi dapat dilihat bahwa inisialisasi *cluster* dengan nilai 3 memberikan waktu komputasi yang lebih cepat jika dibandingkan dengan *cluster* 5 atau *cluster* 7.

Pada citra 42049.jpg nilai *cluster* yang menghasilkan segmentasi terbaik adalah *cluster* dengan nilai 3. Begitu juga

dengan data input citra yang lainnya. Jumlah inisialisasi cluster 3 dianggap paling baik karena menghasilkan nilai kuantitatif segmentasi citra yang paling kecil dibanding dengan *cluster* yang lannya. Selain itu juga memberikan waktu komputasi yang lebih cepat. Hasil segmentasi citra 86000.jpg dapat dilihat pada Gambar 6.8 dan citra 296059.jpg dapat dilihat pada Gambar 6.9. Untuk citra 42049.jpg, 109053.jpg, 147091.jpg, dan 189080.jpg dapat dilihat pada Lampiran Tabel A.4 sampai dengan Tabel A.7.

Tabel 6.9 Hasil Evaluasi Kualitas *Cluster* pada Perbandingan Nilai *Cluster*

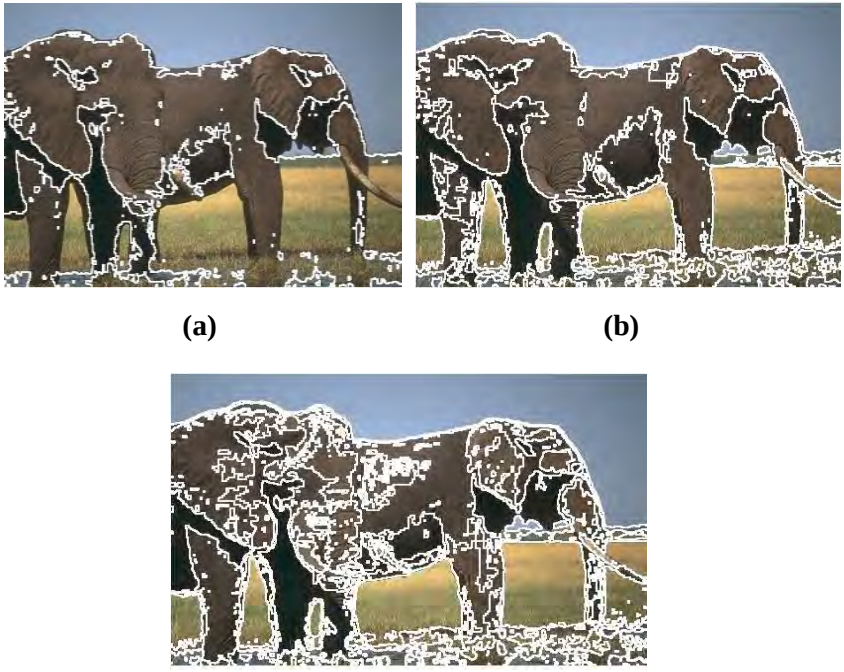
No.	Nama Citra	Jumlah *Cluster*	Waktu (detik)	F(I)
1.	42049.jpg	3	44	**14,577**
		5	48	22,724
		7	57	26,711
2.	86000.jpg	3	102	**315,702**
		5	125	360,787
		7	267	383,251
3.	109053.jpg	3	50	**19,336**
		5	66	23,132
		7	97	29,976
4.	147091.jpg	3	57	**83,275**
		5	63	118,006
		7	78	106,100
5.	189080.jpg	3	54	**35,867**
		5	105	37,459
		7	124	61,589
6.	296059.jpg	3	45	**57,437**
		5	35	95,621
		7	78	110,794

(a)

(b)

(c)

Gambar 6.8 Perbandingan Citra Hasil Segmentasi Berdasarkan Nilai *Cluster* pada Citra 86000.jpg (a) *Cluster* 3 (b) *Cluster* 5 (c) *Cluster* 7



(a) **(b)**

(c)

Gambar 6.9 Perbandingan Citra Hasil Segmentasi Berdasarkan Nilai *Cluster* pada Citra 296059.jpg (a) *Cluster* 3 (b) *Cluster* 5 (c) *Cluster* 7

6.4.3 Uji Kinerja Skenario 2: Perbandingan Hasil Segmentasi Citra Berdasarkan Parameter T_u Awal

Pada skenario ini akan dibandingkan parameter T_u awal yang digunakan dalam proses segmentasi. Seperti telah dijelaskan sebelumnya bahwa nilai T_u ini berpengaruh terhadap nilai dari fungsi keanggotaan objek u_{ci}. Sehingga sangat penting disini untuk mengetahui nilai seberapa berpengaruhnya nilai parameter

awal untuk T_u. Karena *range* T_u yang optimal berkisar antara nilai 1 sampai 30, maka disini diambil nilai 10,20, dan 30 sebagai perbandingan.

Pada citra uji coba dapat dilihat bahwa inisialisasi awal untuk parameter T_u memberikan pengaruh yang cukup signifikan terhadap nilai dari nilai kuantitatif segmentasi citra (F). Hal ini karena parameter T_u berpengaruh terhadap nilai fungsi keanggotaan objek u_{ci}, namun berubahnya nilai inisialisasi parameter T_u di awal tergantung dari citra input.

Berdasarkan hasil uji skenario, dapat dilihat besarnya nilai dari parameter T_u awal yang diberikan untuk proses, memberikan hasil yang berbeda tergantung dari citra masukannya. Hasil segmentasi bisa lebih baik atau lebih jelek jika nilai parameter T_u dikurangi atau ditambahkan. Inisialisasi parameter T_u ini tidak memberikan efek yang signifikan terhadap waktu komputasi, meskipun parameter ini berpengaruh terhadap nilai fungsi keanggotaan objek. Hasil uji skenario ini dapat dilihat pada Tabel 6.10.

Pada citra uji 42049.jpg dan 86000.jpg didapatkan hasil segmentasi yang lebih bagus jika menggunakan parameter T_u sebesar 10. Sedangkan untuk citra 109053.jpg dan 189080.jpg didapatkan hasil segmentasi terbaik dengan T_u sebesar 20. Untuk citra 147091.jpg dan 296059.jpg didapatkan hasil segmentasi yang bagus jika menggunakan parameter T_u awal sebesar 30. Hasil segmentasi citra dapat dilihat pada Gambar 6.10 untuk 86000.jpg dan 296059.jpg pada Gambar 6.11. Untuk hasil segmentasi citra yang lain dapat dilihat pada Lampiran Tabel A.8 sampai Tabel A.11.

Tabel 6.10 Hasil Evaluasi Kuantitatif pada Perbandingan Nilai *Tu*

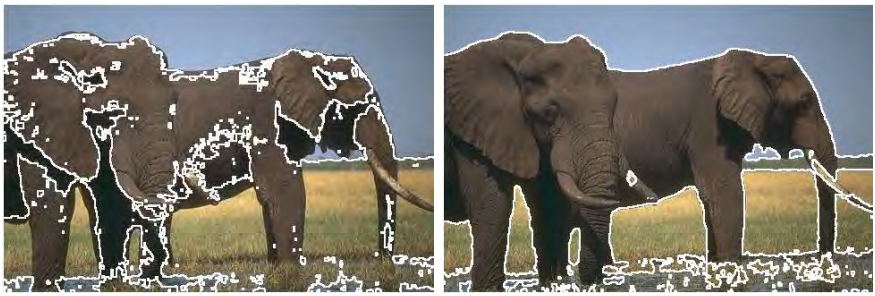
No.	Nama Citra	Tu	Waktu (detik)	Jumlah *Cluster* (C)	F(I)
1.	42049.jpg	10	44	3	18,294
		20	31	5	20,641
		30	14	5	15,902
2.	86000.jpg	10	79	5	318,246
		20	101	3	319,268
		30	110	5	319,188
3.	109053.jpg	10	50	3	19,215
		20	19	2	15,795
		30	22	2	15,807
4.	147091.jpg	10	40	3	91,604
		20	82	4	105,655
		30	32	2	74,944
5.	189080.jpg	10	45	5	40,849
		20	18	5	34,094
		30	21	5	34,753
6.	296059.jpg	10	37	3	56,786
		20	56	3	92,323
		30	44	3	57,009

(a)

(b)

(c)

Gambar 6.10 Hasil Citra Hasil Segmentasi Berdasarkan Nilai Awal Parameter Tupada C itra 86000.jpg a) Tu 10 b) Tu 20 c) Tu 30



(a) **(b)**

(c)

Gambar 6.11 Hasil Citra Hasil Segmentasi Berdasarkan Nilai Awal Parameter Tu pada Citra 296059.jpg
a) Tu 10 b) Tu 20 c) Tu 30

6.4.4 Uji Kinerja Skenario 3: Perbandingan Hasil Segmentasi Citra Berdasarkan Parameter T_v Awal

Skenario ini akan membandingkan hasil segmentasi citra berdasarkan parameter T_v awal yang diinisialisasikan sebelum proses dijalankan. Parameter T_v ini berpengaruh terhadap nilai fungsi keanggotaan fitur v_{cj}. Oleh karena itu, dalam skenario ini bertujuan untuk mengetahui seberapa besar pengaruh T_v terhadap hasil segmentasi citra. Nilai T_v yang optimal berkisar antara nilai

10^6 sampai dengan 10^8. Sehingga dalam skenario ini, sebagai perbandingan diberikan nilai $9x10^6$, $9x10^7$, dan $9x10^8$.

Dari hasil uji skenario, membuktikan bahwa nilai parameter T_v tidak begitu memberikan perbedaan terhadap hasil dari segmentasi citra. Hal ini dikarenakan parameter T_v hanya memberikan sedikit pengaruh terhadap nilai dari fungsi keanggotaan fitur. Dapat dilihat dari hasil uji coba pada Tabel 6.11, bahwa nilai kuantitatif segmentasi citra sebagian besar tidak memiliki selisih yang signifikan antara yang satu dengan yang lainnya.

Citra hasil segmentasi yang diproses dengan perbandingan beberapa nilai T_v tidak menunjukkan perbedaan yang signifikan antara yang satu dengan lainnya. Hal ini membuktikan bahwa parameter T_v memang tidak begitu berpengaruh terhadap hasil segmentasi citra. Namun dari beberapa nilai T_v yang digunakan dapat dilihat bahwa sebagian besar uji skenario yang menggunakan T_v dengan nilai $9x10^7$ memberikan hasil yang lebih baik dibandingkan dengan nilai *Tv* yang lain.

Untuk lebih jelas mengenai hasil dari uji skenario yang dilakukan maka hasil dari uji skenario ditampilkan dalam Tabel 6.9. Sedangkan untuk hasil dari segmentasi citra 189080.jpg dapat dilihat pada Gambar 6.12 dan citra 296059.jpg pada Gambar 6.13. Untuk hasil segmentasi citra yang lainnya dapat dilihat pada Lampiran Tabel A.12 sampai Tabel A.15.

Tabel 6.11 Hasil Evaluasi Kuantitatif pada Perbandingan Nilai *Tv*

No.	Nama Citra	Tv	Waktu (detik)	Jumlah *Cluster* (C)	F(I)
1.	42049.jpg	$9x10^6$	44	3	16,761
		$9x10^7$	42	3	**13,831**
		$9x10^8$	43	3	16,831
2.	86000.jpg	$9x10^6$	246	5	346,232
		$9x10^7$	89	5	**315,273**
		$9x10^8$	104	5	318,246
3.	109053.jpg	$9x10^6$	51	3	19,225
		$9x10^7$	58	3	**18,922**
		$9x10^8$	46	3	19,122
4.	147091.jpg	$9x10^6$	52	5	91,565
		$9x10^7$	45	5	90,277
		$9x10^8$	57	5	**80,468**
5.	189080.jpg	$9x10^6$	15	2	50,221
		$9x10^7$	46	3	**34,213**
		$9x10^8$	80	3	40,757
6.	296059.jpg	$9x10^6$	58	3	92,397
		$9x10^7$	40	3	57,438
		$9x10^8$	37	3	**57,343**

(a)

(b)

(c)

Gambar 6.12 Hasil Citra Hasil Segmentasi Berdasarkan Nilai Awal Parameter Tu pada Citra 86000.jpg
a) Tv $9x10^6$ b) Tv $9x10^7$ c) Tv $9x10^8$

(a)

(b)

(c)

Gambar 6.13 Hasil Citra Hasil Segmentasi Berdasarkan Nilai Awal Parameter Tu pada Citra 296059.jpg
a) Tv $9x10^6$ b) Tv $9x10^7$ c) Tv $9x10^8$

6.4.5 Uji Kinerja Skenario 4: Perbandingan Ukuran Piksel Citra Berwarna

Skenario uji coba ini akan membandingkan hasil segmentasi citra berdasarkan ukuran piksel citra berwarna yang berbeda-beda, yaitu 481x321, 361x241, dan 241x161. Uji coba skenario ini dilakukan terhadap enam citra berwarna

diantaranya adalah 42049.jpg, 86000.jpg, 109053.jpg, 147091.jpg, 189080.jpg, 296059.jpg dan 216081.jpg.

Pada hasil uji coba yang ditunjukkan pada Tabel 6.12 dengan melakukan perbandingan hasil segmentasi berdasarkan ukuran piksel citra yang digunakan untuk uji coba yaitu 481x321, 361x241, dan 241x161 membuktikan bahwa ukuran piksel mempengaruhi hasil segmentasi citra dan lamanya waktu komputasi. Hal ini terjadi karena jika ukuran piksel citra semakin kecil maka secara otomatis banyaknya titik data dalam citra juga akan semakin kecil. Dengan semakin sedikitnya jumlah titik data dalam citra maka waktu yang dibutuhkan untuk komputasi juga akan semakin sedikit. Berubahnya ukuran piksel dari 241x161 ke 361x241 menyebabkan meningkatnya kecepatan komputasi sebesar 42% dan dari piksel 361x241 ke 481x321 memberikan kenaikan sebesar 62%.

Pada citra hasil segmentasi menunjukkan bahwa semua citra dengan ukuran yang paling kecil yaitu 241x161 memiliki nilai kuantatif segmentasi yang paling kecil. Selain itu juga semua citra masukan dengan ukuran piksel yang paling kecil membutuhkan waktu komputasi yang semakin kecil pula. Hasil segmentasi citra dari uji coba skenario ini dapat dilihat lebih jelas pada Gambar 6.14 untuk citra 86000.jpg, sedangkan untuk citra 296059.jpg dapat dilihat pada Gambar 6.15. Untuk hasil segmentasi citra yang lainnya dapat dilihat pada Lampiran Tabel A.16 sampai Tabel A.19.

Tabel 6.12 Hasil Evaluasi Kuantitatif pada Perbandingan Ukuran Piksel Citra

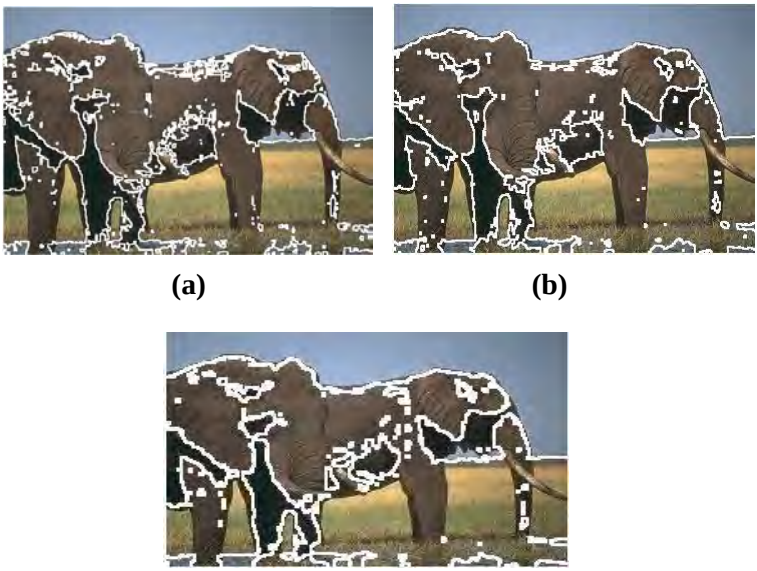
No.	Nama Citra	Ukuran Citra	Waktu (detik)	Jumlah *Cluster*	F(I)
1.	42049.jpg	481x321	41	3	14,589
		361x241	32	3	11,915
		241x161	25	3	**7,851**
2.	86000.jpg	481x321	164	4	332,964
		361x241	61	3	231,848
		241x161	35	6	**153,128**
3.	109053.jpg	481x321	54	3	19,287
		361x241	39	3	13,953
		241x161	29	3	**9,139**
4.	147091.jpg	481x321	43	3	91,616
		361x241	36	3	66,446
		241x161	25	6	**41,918**
5.	189080.jpg	481x321	49	5	39,703
		361x241	36	5	25,213
		241x161	27	5	**19,611**
6.	296059.jpg	481x321	56	3	56,162
		361x241	31	3	42,607
		241x161	25	3	**28,059**

(a)

(b)

(c)

Gambar 6.14 Perbandingan citra segmentasi berdasarkan ukuran piksel citra pada citra 86000.jpg a) Ukuran piksel 321x481 b) Ukuran piksel citra 241x361 c) Ukuran piksel citra 161x241



(a) **(b)**

(c)

Gambar 6.15 Perbandingan citra segmentasi berdasarkan ukuran piksel citra pada citra 296059.jpg a) Ukuran piksel 321x481 b) Ukuran piksel citra 241x361 c) Ukuran piksel citra 161x241

6.4.6 Uji Kinerja Skenario 5: Perbandingan Jumlah Ukuran Bakteri sebagai Ukuran Domain Solusi

Uji kinerja ini akan membandingkan hasil segmentasi citra berdasarkan jumlah bakteri yang digunakan dalam proses optimasi dengan PSO-MPB. Dalam proses optimasi yang digunakan untuk mengoptimalkan parameter T_u dan T_v, jumlah bakteri memiliki peran yang signifikan karena disini bakteri berperan sebagai domain solusi. Sehingga dalam uji coba ini

akan dilihat perbandingan penggunaan bakteri dalam beberapa nilai yaitu, 50, 100, dan 150.

Berdasarkan uji skenario yang dilakukan pada citra berwarna 42049.jpg, 86000.jpg, 109053.jpg, 147091.jpg, 189080.jpg, 296059.jpg dan 216081.jpg didapatkan bahwa semakin besar jumlah bakteri yang digunakan dalam proses optimasi, maka waktu komputasi yang dibutuhkan juga semakin besar. Kenaikan jumlah bakteri dari 50 ke 100 memberikan efek pada penambahan waktu komputasi sebesar 58% dan kenaikan jumlah bakteri dari 100 ke 150 memberikan penambahan waktu komputasi sebesar 51%. Untuk lebih jelas, hasil uji coba ini ditampilkan dalam Tabel 6.13.

Pada beberapa hasil uji coba, citra dengan jumlah bakteri 100 memiliki nilai (F) yang lebih kecil dibandingkan citra dengan jumlah bakteri yang lainnya. Meski kadang hasil yang didapatkan dengan menambahkan jumlah bakteri menjadi lebih bagus, namun waktu komputasi yang dibutuhkan menjadi jauh lebih lama. Untuk citra 42049.jpg dapat dilihat bahwa dengan inisialisasi bakteri sebanyak 50, maka yang dibutuhkan untuk komputasi adalah sebesar 39 detik, sedangkan jika jumlah inisialisasi bakteri dinaikkan menjadi 100, maka waktu yang dibutuhkan adalah 95 detik, yang bisa dikatakan lebih dari dua kali lipat, sedangkan dengan penambahan jumlah bakteri menjadi 150, waktu yang digunakan untuk proses komputasi menjadi 193. Tetapi, dengan adanya kenaikan bakteri pada inisialisasi, nilai dari evaluasi kuantitatif menjadi berkurang yang bisa menjadi indikasi bahwa hasil segmentasi lebih baik.

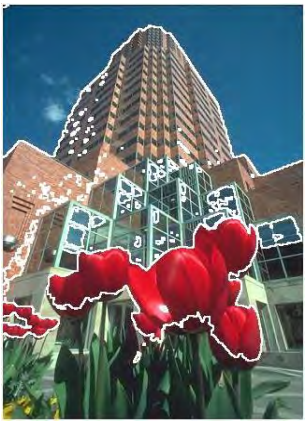
Citra hasil optimasi dengan perbandingan jumlah bakteri ini dapat dilihat pada Gambar 6.16 dan Gambar 6.17.

Sedangkan untuk hasil dari citra yang lain dapat dilihat pada Lampiran Tabel A.20 sampai Tabel A.23.

Tabel 6.13 Hasil Evaluasi Kuantitatif Berdasarkan Perbandingan Jumlah Bakteri yang Digunakan untuk Proses Optimasi

No.	Nama Citra	Jumlah Bakteri	Waktu (detik)	Jumlah *Cluster*	F(I)
1.	42049.jpg	50	39	3	19,349
		100	95	3	18,154
		150	193	3	18,905
2.	86000.jpg	50	85	6	361,465
		100	152	3	317,029
		150	305	4	331,461
3.	109053.jpg	50	44	3	19,351
		100	106	3	18,804
		150	206	3	19,152
4.	147091.jpg	50	50	3	88,677
		100	115	3	91,241
		150	201	6	90,797
5.	189080.jpg	50	39	5	39,515
		100	102	5	35,083
		150	197	5	34.622
6.	296059.jpg	50	93	4	93,241
		100	105	3	57,006
		150	184	3	57.447

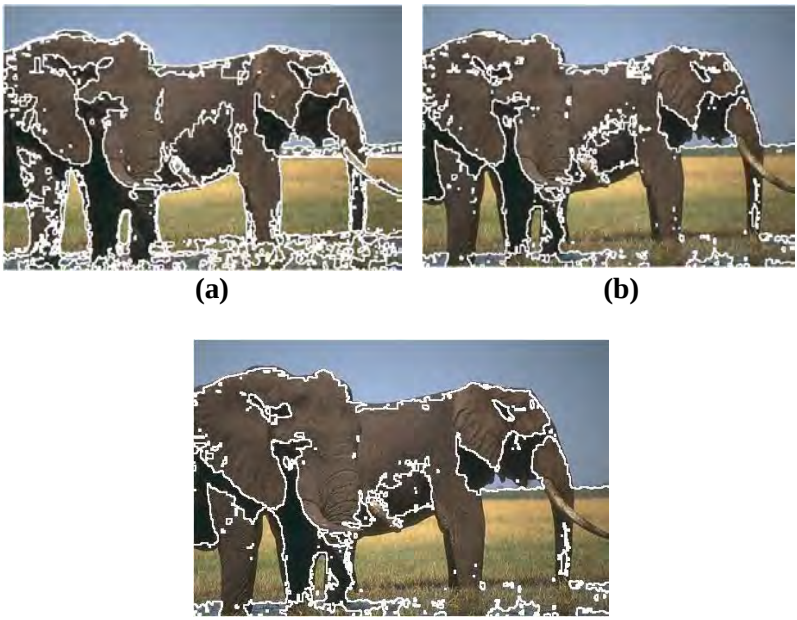
(a)



(b)

(c)

Gambar 6.16 Perbandingan Hasil Segmentasi Citra Berdasarkan Jumlah Bakteri pada Citra 86000.jpg a) Bakteri 50 b) Bakteri 100 c) Bakteri 150



(a) **(b)**

(c)

Gambar 6.17 Perbandingan Hasil Segmentasi Citra Berdasarkan Jumlah Bakteri pada Citra 296059.jpg
a) Bakteri 50 b) Bakteri 100 c) Bakteri 150

6.4.7 Uji Kinerja Skenario 6: Perbandingan Hasil Segmentasi Berdasarkan Jumlah Tahapan Kemotaksis

Pada uji kinerja ini akan membandingkan hasil segmentasi citra berdasarkan tahapan kemotaksis yang dilakukan oleh bakteri dalam melakukan optimasi. Optimasi dengan PSO-MPB ini bertujuan untuk mendapatkan parameter T_u dan T_v yang paling optimal. Dimana T_u dan T_v ini akan memberikan

pengaruh terhadap nilai dari fungsi keanggotaan objek dan fungsi keanggotaan fitur. Jadi, penting disini untuk mengetahui seberapa signifikan pengaruh dari nilai tahapan kemotaksis N_c yang dilakukan oleh bakteri. Nilai dari tahap kemotaksis yang akan digunakan sebagai perbandingan adalah 50, 100, dan 200.

Uji skenario yang dilakukan pada citra berwarna 42049.jpg, 86000.jpg, 109053.jpg, 147091.jpg, 189080.jpg, 296059.jpg dan 216081.jpg memberikan hasil bahwa semakin besar langkah kemotaksis yang dilakukan oleh bakteri dalam proses optimasi, maka waktu yang dibutuhkan untuk melakukan komputasi juga akan semakin besar. Hal ini karena dalam melakukan optimasi, bakteri akan melakukan langkah *swim* dan *tumble* yang merupakan bagian dari kemotaksis sebanyak jumlah *Nc* yang diinisialisasikan. Bertambahnya langkah kemotaksis dari 100 ke 150 memberikan penambahan waktu komputasi sebesar 25%, dan bertambahnya langkah kemotaksis dari 150 ke 200 memberikan penambahan waktu komputasi sebesar 40%.

Selain berpengaruh terhadap waktu komputasi, jumlah tahap kemotaksis juga memberikan pengaruh terhadap hasil dari segmentasi citra. Hal ini terlihat dari jumlah F(I) yang dihasilkan dari tiap proses. Untuk lebih jelasnya hasil skenario ini ditampilkan dalam Tabel 6.14. Hasil segmentasi citra dapat dilihat pada Gambar 6.18 untuk citra 86000.jpg dan Gambar 6.19 untuk citra 296059.jpg. Untuk citra berwarna yang lainnya dapat dilihat di Lampiran Tabel A.24 sampai Tabel A.27 .

Tabel 6.14 Hasil Evaluasi Kuantitatif Berdasarkan Perbandingan Jumlah Tahapan Kemotaksis

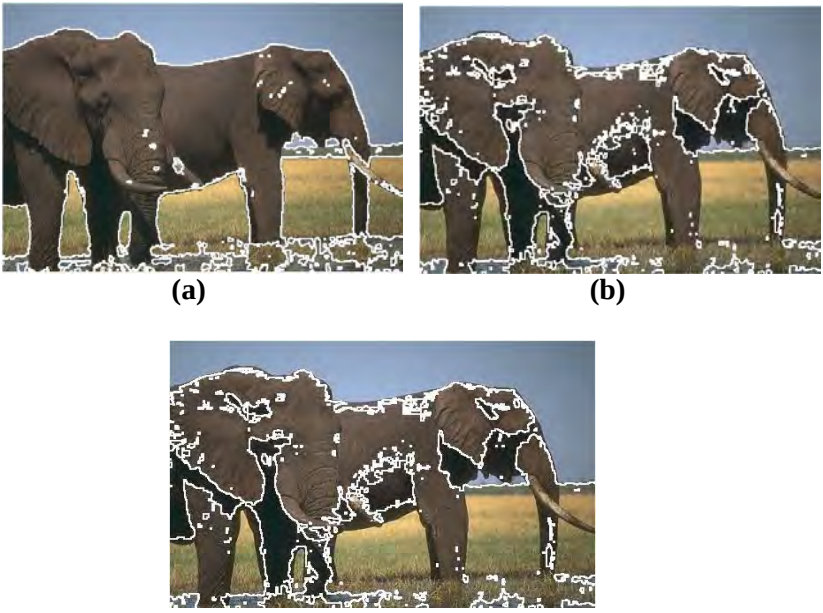
No.	Nama Citra	Nc	Waktu (detik)	Jumlah *Cluster* (C)	F(I)
1.	42049.jpg	50	30	3	19,367
		100	40	3	18,891
		200	67	3	16,646
2.	86000.jpg	50	93	5	319,391
		100	125	5	345.720
		200	263	5	318,022
3.	109053.jpg	50	29	3	15,842
		100	47	3	17,826
		200	64	3	19,240
4.	147091.jpg	50	35	5	87,854
		100	47	5	91,113
		200	69	5	90,012
5.	189080.jpg	50	41	3	39,060
		100	49	3	35.691
		200	55	3	34.094
6.	296059.jpg	50	46	3	84.122
		100	54	3	56,709
		200	57	3	56.937

(a)

(b)

(c)

Gambar 6.18 Perbandingan Hasil Segmentasi Citra Berdasarkan Jumlah Tahap Kemotaksis pada Citra 86000.jpg a) Nc 50 b) Nc 100 c) Nc 200



(a) **(b)**

(c)

Gambar 6.19 Perbandingan Hasil Segmentasi Citra Berdasarkan Jumlah Tahap Kemotaksis pada Citra 296059.jpg a) Nc 50 b) Nc 100 c) Nc 200

6.4.8 Evaluasi Terhadap Uji Coba Segmentasi Citra dengan FCCI

Pada uji coba yang pertama, yaitu uji kebenaran terhadap metode optimasi PSO-MPB dilakukan terhadap enam fungsi nonlinear. Pengujian ini bertujuan untuk mengetahui efisiensi hasil implementasi algoritma PSO-MPB dalam menyelesaikan sistem persamaan nonlinear. Pada uji coba ini, pengujian dianggap berhasil apabila hasil pengujian sama atau mendekati hasil asli fungsi.

Berdasarkan enam uji kebenaran yang dilakukan terhadap PSO-MPB, hampir semua hasil uji coba memberikan hasil yang sama dengan hasil fungsi aslinya. Oleh karena itu, dapat disimpulkan bahwa algoritma PSO-MPB mempunyai kinerja yang baik dalam mencari solusi optimal fungsi objektif.

Uji coba kedua yang dilakukan terhadap proses segmentasi citra dengan menggunakan enam citra masukan dan beberapa parameter memberikan hasil yang secara garis besar dapat dilihat pada beberapa pembahasan berikut ini.

Berdasarkan uji coba skenario yang pertama yang dilakukan terhadap enam citra masukan dengan parameter inisialisasi *cluster* awal sebesar 3,5, dan 7 memberikan hasil bahwa semakin besar jumlah *cluster* awal, waktu komputasi yang dibutuhkan juga akan semakin besar. Kenaikan jumlah *cluster* dari 3 ke 5 memberikan kenaikan waktu komputasi sebesar 8% dan kenaikan dari *cluster* 5 ke *cluster* 7 memberikan kenaikan sebesar 15%. Selain itu, nilai kuantitatif segmentasi citra (F) juga akan meningkat jika jumlah *cluster* awal semakin besar.

Uji skenario kedua yang dilakukan dengan memberikan perbedaan pada nilai awal parameter T_u yaitu 10, 20, dan 30. Nilai ini dipilih karena rentang optimal parameter T_u untuk fungsi objektif dari FCCI adalah antara 1 sampai dengan 30. Berdasarkan hasil uji coba dapat diketahui bahwa inisialisasi awal untuk parameter T_u memberikan pengaruh yang cukup signifikan terhadap nilai dari nilai kuantitatif segmentasi citra (F). Hal ini karena parameter T_u berpengaruh terhadap nilai fungsi keanggotaan objek u_{ci}, namun berubahnya nilai inisialisasi parameter T_u di awal tergantung dari citra input.

Berdasarkan hasil uji skenario, dapat dilihat besarnya nilai dari parameter T_u awal yang diberikan untuk proses,

memberikan hasil yang berbeda tergantung dari citra masukannya. Hasil segmentasi bisa lebih baik atau lebih jelek jika nilai parameter T_u dikurangi atau ditambahkan. Inisialisasi parameter T_u ini tidak memberikan efek yang signifikan terhadap waktu komputasi, meskipun parameter ini berpengaruh terhadap nilai fungsi keanggotaan objek.

Kemudian untuk uji skenario berdasarkan pada parameter T_v dengan nilai parameter T_v yang digunakan adalah $9x10^6$, $9x10^7$, dan $9x10^8$, memberikan hasil bahwa parameter T_v tidak begitu memberikan efek yang signifikan terhadap hasil segmentasi citra. Hal ini bisa dilihat dari nilai kuantitatif segmentasi citra dengan Liu dan Yang's (F) yang tidak begitu berbeda antara nilai T_v yang satu dengan T_v yang lainnya dalam satu citra. Namun, berdasarkan hasil uji coba, sebagian besar citra dengan parameter T_v sebesar $9x10^7$ memiliki nilai (F) lebih kecil dibanding dengan citra yang menggunakan nilai parameter T_v yang lain.

Selanjutnya, uji coba terhadap ukuran piksel terhadap hasil segmentasi citra. Ukuran piksel yang digunakan adalah 481x321, 361x241, dan 241x161. Dari hasil uji coba ini, dapat dibuktikan bahwa citra dengan ukuran piksel terkecil membutuhkan waktu komputasi yang lebih cepat dan menghasilkan nilai (F) yang lebih kecil pula. Cepatnya waktu komputasi ini dikarenakan jumlah piksel dalam citra yang semakin sedikit, sehingga waktu yang dibutuhkan untuk memprosesnya semakin sedikit pula. Berubahnya ukuran piksel dari 241x161 ke 361x241 menyebabkan meningkatnya kecepatan komputasi sebesar 42% dan dari piksel 361x241 ke 481x321 memberikan kenaikan sebesar 62%.

Uji coba yang kelima dilakukan untuk mengetahui pengaruh jumlah bakteri yang digunakan terhadap hasil

segmentasi citra. Jumlah bakteri yang digunakan adalah 50,100, dan 150. Dari hasil uji coba didapatkan bahwa semakin besar jumlah bakteri yang digunakan dalam proses optimasi, maka waktu komputasi yang dibutuhkan juga semakin besar. Kenaikan jumlah bakteri dari 50 ke 100 memberikan efek pada penambahan waktu komputasi sebesar 58% dan kenaikan jumlah bakteri dari 100 ke 150 memberikan penambahan waktu komputasi sebesar 51%. Pada beberapa hasil uji coba, citra dengan jumlah bakteri 100 memiliki nilai (F) yang lebih kecil dibandingkan citra dengan jumlah bakteri yang lainnya.

Skenario uji coba terakhir untuk uji coba segmentasi ini adalah untuk mengetahui pengaruh jumlah tahapan kemotaksis terhadap hasil segmentasi. Uji coba ini memberikan hasil bahwa semakin besar langkah kemotaksis yang dilakukan oleh bakteri dalam proses optimasi, maka waktu yang dibutuhkan untuk melakukan komputasi juga akan semakin besar. Bertambahnya langkah kemotaksis dari 100 ke 150 memberikan penambahan waktu komputasi sebesar 25%, dan bertambahnya langkah kemotaksis dari 150 ke 200 memberikan penambahan waktu komputasi sebesar 40%. Hal ini karena dalam melakukan optimasi, bakteri akan melakukan langkah *swim* dan *tumble* yang merupakan bagian dari kemotaksis sebanyak jumlah *Nc* yang diinisialisasikan. Sehingga secara otomatis, jika jumlah tahapan kemotaksis ditingkatkan, maka waktu yang dibutuhkan juga akan meningkat.

BAB VII
KESIMPULAN DAN SARAN

Pada bab ini dijelaskan mengenai kesimpulan yang didapat setelah melakukan serangkaian uji coba. Selain itu, dalam bab ini dijelaskan pula mengenai saran-saran untuk pengembangan perangkat lunak lebih lanjut.

7.1 Kesimpulan

Kesimpulan yang diperoleh berdasarkan uji coba dan evaluasi yang telah dilakukan adalah sebagai berikut:

1. Algoritma *Fuzzy Co-Clustering For Images* (FCCI) dapat digunakan untuk melakukan proses segmentasi citra dengan hasil yang optimal.
2. Algoritma *Particle Swarm Optimization* dengan Modifikasi Perilaku Bakteri (PSO-MPB) secara optimal melakukan optimasi terhadap parameter *threshold* yaitu T_u dan T_v.
3. Hasil segmentasi citra dengan menggunakan FCCI dan PSO-MPB menunjukkan hasil segmentasi citra yang baik dengan menghasilkan evaluasi kuantitatif yang relatif kecil dan rata-rata *error color* kurang dari 0,1%.
4. Semakin kecil ukuran piksel citra yang digunakan maka waktu komputasi yang diperlukan juga semakin cepat, begitu pula sebaliknya.
5. Inisialisasi *cluster* pada awal proses memberikan efek pada kinerja algoritma dengan memberikan efek pada meningkatnya waktu komputasi sebesar 8% untuk kenaikan jumlah *cluster* dari 3 ke 5 dan kenaikan dari *cluster* 5 ke *cluster* 7 memberikan kenaikan sebesar 15%.
6. Inisialisasi awal untuk parameter T_u memberikan pengaruh yang cukup signifikan terhadap nilai dari nilai kuantitatif segmentasi

citra (F) namun tidak memberikan efek yang cukup besar terhadap waktu komputasi.

7. Inisialisasi parameter T_v tidak begitu memberikan efek yang signifikan terhadap hasil segmentasi citra, tetapi parameter T_v sebesar $9x10^7$ menghasilkan nilai (F) lebih kecil dibanding dengan citra yang menggunakan nilai parameter T_v yang lain.
8. Ukuran piksel memberikan efek yang signifikan terhadap waktu komputasi.
9. Kenaikan jumlah bakteri pada proses optimasi memberikan efek terhadap lamanya waktu komputasi. Tetapi, peningkatan lamanya waktu komputasi ini disertai dengan hasil citra yang lebih optimal.
10. Inisialisasi jumlah tahapan kemotaksis memberikan efek terhadap waktu komputasi dengan kenaikan waktu komputasi antara 25% sampai dengan 40%.
11. Citra hasil uji coba dengan metode FCCI yang memiliki nilai *F* lebih kecil dari Madasu Hanmandlu *et al* sebesar 64% dan 34% memiliki nilai *F* yang lebih besar.

7.2 Saran

Beberapa saran yang hendak disampaikan terkait dengan pengerjaan tugas akhir ini adalah:

1. Perlu dikembangkan metode yang dapat mengeneralisasi parameter-parameter yang dibutuhkan secara otomatis, sehingga pengguna tidak perlu melakukan pencarian nilai parameter yang dapat menghasilkan jumlah *cluster* yang tepat.
2. Perlu dilakukan uji coba yang lebih variatif dan mendalam untuk mengetahui hasil segmentasi citra yang lebih optimal berdasarkan metode FCCI.

LAMPIRAN

Tabel A.1 Citra Berwarna Untuk Uji Coba Kinerja (Bagian 1)

No.	Citra Berwarna
1.	 **42049.jpg**
2.	 **86000.jpg**

Tabel A.2 Citra Berwarna Untuk Uji Coba Kinerja (Bagian 2)

No.	Citra Berwarna
1.	 **109053.jpg**
2.	 **147091.jpg**

Tabel A.3 Citra Berwarna Untuk Uji Coba Kinerja (Bagian 2)

No.	Citra Berwarna
1.	 **189080.jpg**
2.	 **296059.jpg**

Tabel A.4 Perbandingan Citra Hasil Segmentasi Berdasarkan Nilai *Cluster* pada Citra 42049.jpg

	
Cluster = 3 F(I)= 14,577	*Cluster*=5 F(I)= 22,724
	
Cluster=7 F(I)= 26,711	

Tabel A.5 Perbandingan Citra Hasil Segmentasi Berdasarkan Nilai *Cluster* pada Citra 109053.jpg

	
Cluster = 3 F(I)= 19,336	*Cluster*=5 F(I)= 23,132

Cluster=7
F(I)= 29,976

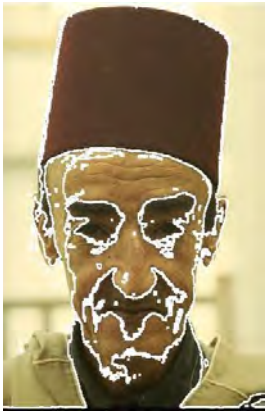
Tabel A.6 Perbandingan Citra Hasil Segmentasi Berdasarkan Nilai *Cluster* pada Citra 147091.jpg

Cluster = 3
F(I)= 83,275

Cluster=5
F(I)= 118,006

Cluster=7
F(I)= 106,100

Tabel A.7 Perbandingan Citra Hasil Segmentasi Berdasarkan Nilai *Cluster* pada Citra 189080.jpg

 *Cluster* = 3 F(I)= 35,867	 *Cluster*=5 F(I)= 37,459
 *Cluster*=7 F(I)= 61,589	

Tabel A.8 Perbandingan Citra Hasil Segmentasi Berdasarkan Parameter *Tu* pada Citra 42049.jpg

 *Tu*= 10 F(I)= 18,294	 *Tu*=20 F(I)= 20,641
 *Tu*=30 F(I)= 15,902	

Tabel A.9 Perbandingan Citra Hasil Segmentasi Berdasarkan Parameter *Tu* pada Citra 109053.jpg

 *Tu*= 10 F(I)= 19,215	 *Tu*=20 F(I)= 15,795

Tu=30
F(I)= 15,807

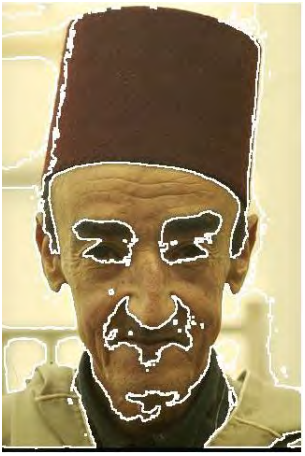
Tabel A.10 Perbandingan Citra Hasil Segmentasi Berdasarkan Parameter *Tu* pada Citra 147091.jpg

Tu= 10
F(I)= 91,604

Tu=20
F(I)= 105,655

Tu=30
F(I)= 74,944

Tabel A.11 Perbandingan Citra Hasil Segmentasi Berdasarkan Parameter *Tu* pada Citra 189080.jpg

 *Tu* = 10 F(I)= 40,849	 *Tu*=20 F(I)= 34,094
Tu=30  F(I)= 34,753	

Tabel A.12 Perbandingan Citra Hasil Segmentasi Berdasarkan Parameter *Tv* pada Citra 42049.jpg

	
Tv= 9x10^6 F(I)= 16,761	*Tv*= 9x10^7 F(I)= 13,831
 *Tv*= 9x10^8 F(I)= 14,38	

Tabel A.13 Perbandingan Citra Hasil Segmentasi Berdasarkan Parameter *Tv* pada Citra 109053.jpg

	
Tv= 9x10^6 F(I)= 19,225	*Tv*= 9x10^7 F(I)= 18,922

Tv= 9x10^{8}
F(I)= 19,122

Tabel A.14 Perbandingan Citra Hasil Segmentasi Berdasarkan Parameter *Tv* pada Citra 147091.jpg

Tv= 9x10^{6}
F(I)= 91,565

Tv= 9x10^{7}
F(I)= 90,277

Tv= 9x10^{8}
F(I)= 80,468

Tabel A.15 Perbandingan Citra Hasil Segmentasi Berdasarkan Parameter *Tv* pada Citra 189080.jpg

	
$Tv = 9x10^6$ F(I)= 50,221	$Tv=9x10^7$ F(I)= 34,213
	
$Tv=9x10^8$ F(I)= 40,757	

Tabel A.16 Perbandingan Citra Hasil Segmentasi Berdasarkan Ukuran Piksel pada Citra 42049.jpg

 Ukuran= 481x321 F(I)= 14,589	 Ukuran= 361x241 F(I)= 11,915
 Ukuran= 241x161 F(I)= 7,851	

Tabel A.17 Perbandingan Citra Hasil Segmentasi Berdasarkan Ukuran Piksel pada Citra 109053.jpg

 Ukuran= 481x321 F(I)= 19,287	 Ukuran= 361x241 F(I)= 13,953

Ukuran= 241x161
F(I)= 9,139

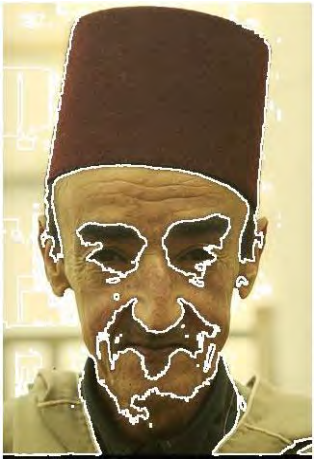
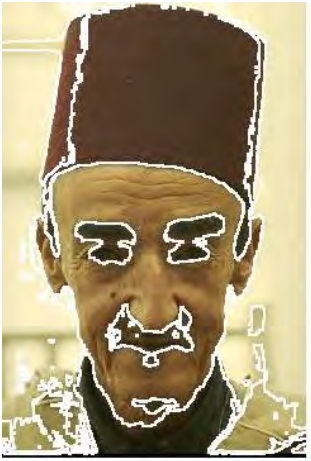
Tabel A.18 Perbandingan Citra Hasil Segmentasi Berdasarkan Ukuran Piksel pada Citra 147091.jpg

Ukuran= 481x321
F(I)= 91,616

Ukuran= 361x241
F(I)= 66,446

Ukuran= 241x161
F(I)= 41,918

Tabel A.19 Perbandingan Citra Hasil Segmentasi Berdasarkan Ukuran Piksel pada Citra 189080.jpg

 Ukuran= 481x321 F(I)= 39,703	 Ukuran= 361x241 F(I)= 25,213
 Ukuran= 241x161 F(I)= 19,611	

Tabel A.20 Perbandingan Citra Hasil Segmentasi Berdasarkan Jumlah Bakteri pada Citra 42049.jpg

 Bakteri= 50 F(I)= 19,349	 Bakteri= 100 F(I)= 18,154
 Bakteri= 150 F(I)= 18,905	

Tabel A.21 Perbandingan Citra Hasil Segmentasi Berdasarkan Jumlah Bakteri pada Citra 109053.jpg

 Bakteri= 50 F(I)= 19,351	 Bakteri= 100 F(I)= 18,804

Bakteri= 150
F(I)= 19,152

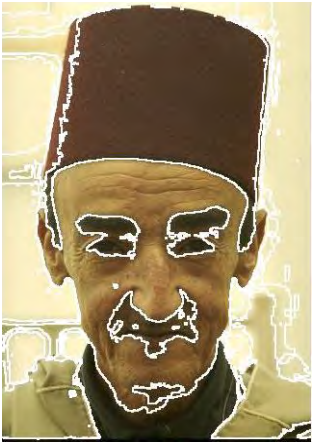
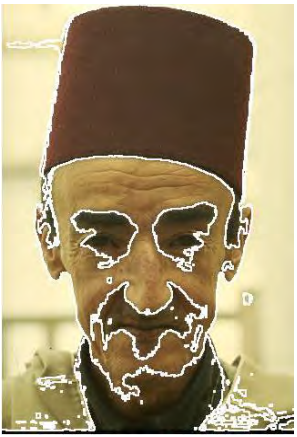
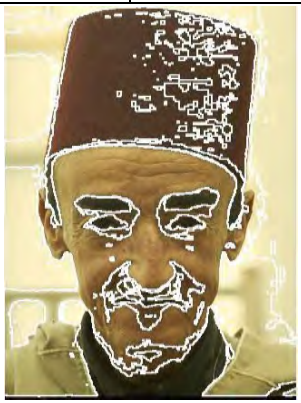
Tabel A.22 Perbandingan Citra Hasil Segmentasi Berdasarkan Jumlah Bakteri pada Citra 147091.jpg

Bakteri= 50
F(I)= 88,677

Bakteri= 100
F(I)= 91,241

Bakteri= 150
F(I)= 90,797

Tabel A.23 Perbandingan Citra Hasil Segmentasi Berdasarkan Jumlah Bakteri pada Citra 189080.jpg

	
Bakteri= 50 F(I)= 39,515	Bakteri= 100 F(I)= 35,083
	
Bakteri= 150 F(I)= 34,622	

Tabel A.24 Perbandingan Citra Hasil Segmentasi Berdasarkan Tahapan Kemotaksis pada Citra 42047.jpg

 Nc= 50 F(I)= 19,367	 Nc= 100 F(I)= 18,891
 Nc= 200 F(I)= 16,646	

Tabel A.25 Perbandingan Citra Hasil Segmentasi Berdasarkan Tahapan Kemotaksis pada Citra 109053.jpg

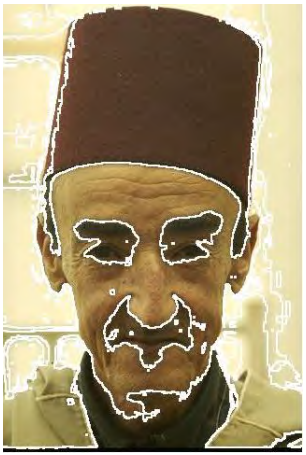
 Nc= 50 F(I)= 15,842	 Nc= 100 F(I)= 17,826

 Nc= 200 F(I)= 19,240

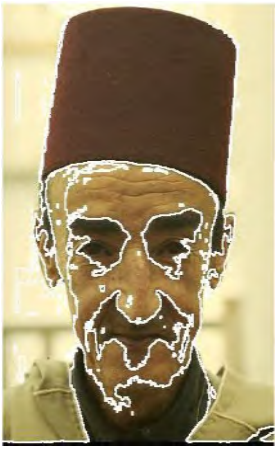
Tabel A.26 Perbandingan Citra Hasil Segmentasi Berdasarkan Tahapan Kemotaksis pada Citra 147091.jpg

 Nc= 50 F(I)= 87,854	 Nc= 100 F(I)= 91,113
 Nc= 200 F(I)= 90,012	

Tabel A.27 Perbandingan Citra Hasil Segmentasi Berdasarkan Tahapan Kemotaksis pada Citra 189080.jpg



Nc= 50
F(I)= 39,060



Nc= 100
F(I)= 35,691

Nc= 200
F(I)= 34,094

Tabel A.28 Hasil Uji Kebenaran Segmentasi Citra (Bagian 1)

No.	Nama Citra	S		*Cluster*		F	
		1	2	1	2	1	2
1.	42049.jpg	332,28	324,12	3	3	19,212	18,814
2	86000.jpg	54,124	58,689	3	3	329,123	315,702
3.	109053.jpg	89,324	78,890	3	3	19,255	19,336
4.	147091.jpg	70,567	80,321	3	3	84,236	83,275
5.	296059.jpg	56,987	57,432	3	3	58,897	57,437
6.	3096.jpg	87,232	51,257	2	2	10,923	4,429
7.	8023.jpg	89,876	78,458	4	4	17,253	16,103
8.	12084.jpg	482,29	490,457	3	3	185,568	184,459
9.	14037.jpg	58,768	57,298	3	3	66,459	64,832
10.	16077.jpg	93,684	94,987	4	4	64,348	67,521
11.	19021.jpg	2937,6	2848,34	3	3	61,985	58,032
12.	21077.jpg	1532,9	1459,2	3	3	15,783	14,885
13.	24077.jpg	124,8	113,2	3	3	43,291	39,327

Tabel A.29 Hasil Uji Kebenaran Segmentasi Citra (Bagian 2)

No.	Nama Citra	S		*Cluster*		F	
		1	**2**	**1**	**2**	**1**	**2**
14.	33039.jpg	730,13	748,9	2	2	7,892	6,206
15.	37073.jpg	267,83	248,5	3	3	35,973	36,733
16.	38082.jpg	297,74	228,8	3	3	333,965	32,211
17.	38092.jpg	895,78	959,09	3	3	66,583	66,464
18.	41033.jpg	297,54	254,73	2	2	15,98	14,72
19.	41069.jpg	712,35	699,43	3	3	12,897	12,636
20.	42012.jpg	250,45	279,3	2	2	10,67	4,87
21.	43074.jpg	402,26	414,13	3	3	32,87	31,500
22.	45096.jpg	300,37	329,73	5	5	81,392	86,453
23.	54082.jpg	187,98	193,09	3	3	64,39	72,31
24.	65033.jpg	322,14	277	4	4	178,93	164,32
25.	66053.jpg	87,93	90,73	3	3	41,983	39,814
26.	86016.jpg	129,23	131,95	2	2	7,867	4,221

Tabel A.30 Hasil Uji Kebenaran Segmentasi Citra (Bagian 3)

No.	Nama Citra	S		*Cluster*		F	
		1	2	1	2	1	2
27.	97033.jpg	479,89	482,74	3	3	37,93	29,69
28.	101087.jpg	89,36	77,56	3	3	100,78	117,72
29.	108005.jpg	299,56	330,93	3	3	45,97	42,42
30.	108082.jpg	289,54	277,55	4	4	20,578	19,447
31.	119082.jpg	104,78	99,62	4	4	31,876	30,828
32.	143090.jpg	300,34	313,77	3	3	59,359	52,203
33.	145086.jpg	80,266	76,588	4	4	300,36	293,48
34.	148026.jpg	60,98	55,73	3	3	90,264	91,212
35.	148089.jpg	212,78	193,6	2	2	59,983	50,940
36.	156065.jpg	508,89	493,13	3	3	129,34	122,17
37.	157055.jpg	419,36	398,40	5	5	60,457	61,604
38.	159008.jpg	204,76	220,96	3	3	63,976	59,219
39.	160068.jpg	100,58	103,70	3	3	25,973	26,031

Tabel A.31 Hasil Uji Kebenaran Segmentasi Citra (Bagian 4)

No.	Nama Citra	S		*Cluster*		F	
		1	2	1	2	1	2
40.	197017.jpg	99,97	94,05	4	4	33,892	34,491
41.	208001.jpg	513,89	493,43	5	5	98,974	104,73
42.	216081.jpg	240,45	230,6	4	4	223,93	225,92
43.	219090.jpg	288,93	259,33	4	4	17,896	14,869
44.	220075.jpg	329,58	339,94	3	3	90,593	97,615
45.	253027.jpg	200,89	208,82	3	3	25,995	26,664
46.	253055.jpg	125,89	118,87	3	3	187,74	193,83
47.	291000.jpg	320,89	353,67	5	5	268,48	271,68
48.	300091.jpg	200,18	219,19	3	3	44,952	43,946
49.	78004.jpg	220,69	237,68	3	3	21,954	23,146
50.	101085.jpg	249,59	236,33	3	3	51,895	52,124

Keterangan : 1 : Hasil berdasarkan paper rujukan

2 : Hasil berdasarkan uji coba

Tabel A.32 Hasil Uji Citra terhadap Nilai *Threshold* Optimal

No.	Nama Citra	Inisialisasi		Optimal		Waktu (detik)	F(I)
		Tu	Tv	Tu	Tv		
1.	109053.jpg	10	$9x10^7$	**10**	**$9x10^7$**	15	25,933
2.	143090.jpg	10	$9x10^7$	**20**	**$9,9x10^7$**	67	52,208
3.	145086.jpg	10	$9x10^7$	**24**	**$9,8x10^7$**	53	273,1
4.	101087.jpg	10	$9x10^7$	**18**	**$9,6x10^7$**	69	117,256
5.	42049.jpg	10	$9x10^7$	**28**	**$9,9x10^7$**	46	18,175
6.	37073.jpg	10	$9x10^7$	**14**	**$9,9x10^7$**	57	35,999
7.	14037.jpg	10	$9x10^7$	**21**	**$9,8x10^7$**	68	59,565
8.	38092.jpg	10	$9x10^7$	**7**	**$9,9x10^7$**	124	106,015
9.	65033.jpg	10	$9x10^7$	**12**	**$9,9x10^7$**	91	165,275
10.	156065.jpg	10	$9x10^7$	**23**	**$9,9x10^7$**	105	123,091
11.	148026.jpg	10	$9x10^7$	**30**	**$9,8x10^7$**	94	94,535
12.	260058.jpg	10	$9x10^7$	**30**	**$9,8x10^7$**	122	155,994
13.	361010.jpg	10	$9x10^7$	**17**	**$9,6x10^7$**	54	57,747
14.	101085.jpg	10	$9x10^7$	**25**	**$9,8x10^7$**	93	143,23

BIODATA PENULIS

Munawaroh, biasa dipanggil Muna, lahir di Tuban pada tanggal 19 September 1992, merupakan anak tunggal. Penulis telah menempuh pendidikan mulai dari SD Negeri Genaharjo II (1998-2004), SMP Negeri 1 Semanding (2004-2007), SMA Negeri I Tuban (2007-2010), dan terakhir Teknik Informatika ITS (2010-2014). Selama kuliah, penulis pernah menjadi asisten pada mata kuliah Sosio Etika dan Kecerdasan Komputasional. Selain itu, penulis juga aktif di organisasi Himpunan Mahasiswa Teknik Computer-Informatika (HMTC) ITS. Dalam menyelesaikan kuliahnya, penulis mengambil bidang minat Komputasi Cerdas dan Visualisasi (KCV) dan tertarik pada hal yang berhubungan dengan pengolahan citra dan kecerdasan buatan. Selain dalam bidang akademik maupun organisasi, penulis sangat suka menghabiskan waktu untuk membaca buku yang berkaitan dengan politik, hukum dan sastra. Selain itu, penulis menggunakan waktu luangnya untuk berolahraga dan pergi jalan-jalan bersama teman-temannya dan juga keluarga. Penulis dapat dihubungi melalui *e-mail*: muna1931@gmail.com.