ITS
Institut
Teknologi
Sepuluh Nopember

TESIS

PENENTUAN TITIK PUSAT AWAL KLASTER PADA ALGORITMA K-MEANS MENGGUNAKAN TITIK REPRESENTASI BERBASIS KEPADATAN

CITRA LESTARI
5105201012

DOSEN PEMBIMBING
Prof. Dr. Ir. Arif Djunaidy, M.Sc.

PROGRAM MAGISTER
BIDANG KEAHLIAN TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOVEMBER
SURABAYA
2008

Tesis disusun untuk memenuhi salah satu syarat memperoleh gelar

Magíster Teknik Informatika (M.Kom)

di

Institut Teknologi Sepuluh November

oleh:

Citra Lestari

Nrp. 5105201012

Tanggal Ujian : 5 Februari 2008

Periode Wisuda : Maret 2008

Disetujui oleh :

1. Prof. Dr. Ir. Arif Djunaidy, M.Sc. (Pembimbing)
   NIP: 131.633.403
   .....

2. Ir. F. X. Arunanto, M.Sc (Penguji)
   NIP: 131.285.53
   .....

3. Dr. Ir. Joko Lianto Buliali, M.Sc (Penguji)
   NIP: 131.996.151
   .....

4. Daniel Oranova Siahaan, S.Kom, M.Sc, PD.Eng (Penguji)
   NIP: 132.318.629
   .....

Direktur Program Pascasarjana,

Prof. Dr. Ir. Suparno, MSIE
NIP. 130.532.035

LEMBAR PERSEMBAHAN

Untuk :
Papa ...
Mama ...
dan diriku sendiri.

Ucapan terima kasih :

- Yesusku, untuk *"taking my wheel"*
- Bunda Maria, untuk doa dan bimbingannya.
- Papa & Mama, untuk semua dukungan, doa, dan cinta
- Prof. Dr. Ir. Arif Djunaidy, M.Sc, untuk waktu, perhatian, dan kesabaran dalam membimbing saya yang 'lemot' ini
- Rifqy Zainal, teman seperjuanganku. Terima kasih karena bersedia jadi *pioneer*-ku ☺
- Para dosen penguji, yang rela menguji saya jam 16.00
- Teman-teman IFT UC dan Kajur saya, yang membolehkan saya 'cabut' pada jam kantor buat mengurus buku ini.
- Teman-teman Eliata, yang telah membuat saya tetap waras dengan ke'gila'an kalian ☺

PENENTUAN TITIK PUSAT AWAL KLASTER PADA ALGORITMA K-MEANS MENGGUNAKAN TITIK REPRESENTASI BERBASIS KEPADATAN

Nama Mahasiswa : Citra Lestari
NRP : 5105201012
Pembimbing : Prof. Dr. Ir. Arif Djunaidy, M.Sc.

ABSTRAK

Algoritma *k*-Means melakukan klasterisasi dengan meletakkan titik data ke dalam klaster yang titik pusatnya berjarak terdekat. Karenanya, kualitas algoritma *k*-Means sangat bergantung pada pemilihan k titik pusat awal klaster. Telah banyak penelitian dilakukan untuk memperbaiki akurasi atau kecepatan algoritma *k*-Means. Salah satunya adalah algoritma *k*-Means sederhana yang membagi set data menjadi beberapa blok per dimensi dan merepresentasikannya dalam sebuah titik representasi. Cara tersebut memang sederhana dan cepat, namun dapat mengakibatkan hilangnya korelasi antar dimensi dalam set data.

Titik representasi dalam algoritma penelitian ini dibuat dengan pendekatan berbasis kepadatan. Awalnya algoritma mencari titik inti dalam set data, lalu mengelompokkan titi*k*-titik inti yang berjangkauan kepadatan satu sama lain kedalam sebuah grup. Setiap grup diwakili oleh titik representasi yang berada di tengah-tengah grup. Algoritma kemudian melakukan klasterisasi berbasis *k*-Means pada titik-titik representasi untuk menentukan titik pusat awal klaster. Karena titik representasi mewakili titik inti dan berada di tengah-tengah titik inti, maka besar kemungkinan titik pusat awal klaster yang dihasilkan telah berada di tengah klaster. Titik pusat awal kalster itu digunakan untuk klasterisasi semua titik data.

Hasil uji coba menunjukkan bahwa kinerja algoritma dalam penelitian ini bergantung pada batas kepadatan. Hasil kinerja optimal diperoleh pada saat beberapa set data dalam uji coba diberikan batas kepadatan 4 dan 17. Untuk penentuan kedua batas kesalahan ini dihasilkan kesalahan klasterisasi yang relatif rendah dan waktu komputasi yang relatif cepat. Selain itu algoritma yang dikembangkan dalam penelitian ini juga memberikan akurasi klasterisasi yang lebih tinggi daripada algoritma pembandingnya.

Kata Kunci : *k*-Means, titik pusat klaster, titik representasi berbasis kepadatan, titik inti

CENTROID INITIALIZATION ON K-MEANS ALGORITHM USING DENSITY BASED REPRESENTATION POINT

Nama Mahasiswa : Citra Lestari
NRP : 5105201012
Pembimbing : Prof. Dr. Ir. Arif Djunaidy, M.Sc.

ABSTRACT

K-Means algorithm clusters data by assigning each point to a cluster having the closest centroid to that point. Therefore, the quality of *k*-means algorithm is very dependent on the selection of initial centroid of the clusters. Many research works have been done to improve the accuracy of *k*-means algorithm. One of them is a simple partition *k*-Means algorithm. It splits data on each dimension into a number of blocks, where each block is then represented by a representation point. Although this method is simple and fast, it could produce loose correlation among dimensions in data set.

In this research, an improvement to the simple partition *k*-Means algorithm is performed by employing a density based approach for determining the representation points. Firstly, the algorithm finds core points from the dataset, which is then continued by classifying core points having density that are reachable to another into a group. Each group is represented by a representation point located in the midst of the group. Secondly, the algorithm performs *k*-Means based clusterization process using the representation points to determine initial centroid of clusters. Since each representation point represented a core points and located in the midst of a core point of each group, there is a great possibility that the initial centroids that have been produced are already positioned in the center of each cluster. Finally, these initial centroids are then used to cluster the whole data set.

Experimental results show that the performance of the algorithm developed in this research is dependent on the minimum threshold of the density. Using the available experiment data sets, the algorithm seems working optimally for the minimum threshold of the density values of 4 and 17. For both of these values, low clustering errors are produced in relatively fast computing time. In addition, the algorithm is also capable of giving better clusterization accuracy in compared to the simple partition *k*-Means algorithm.

Keywords: simple partition *k*-Means algorithm, representation point, density based approach

DAFTAR ISI

DAFTAR GAMBAR

Halaman

DAFTAR ALGORITMA

Halaman

DAFTAR TABEL

Halaman

DAFTAR SEGMEN PROGRAM

Halaman

BAB 1

PENDAHULUAN

1.1 Latar Belakang

K-Means adalah metode klasterisasi terpartisi yang berbasis prototype, yaitu memilah titik-titik data ke dalam kelompok-kelompok (klaster) yang memiliki titik pusat (*centroid*) paling dekat (atau paling similar) dengan masing-masing titik data. Metode ini sangat terkenal dan banyak digunakan dalam aplikasi-aplikasi klasterisasi. Hal tersebut disebabkan oleh algoritma *k*-Means yang terbilang sederhana dan mudah diimplementasikan. Dengan menggunakan perhitungan Euclidean, K-Means mengukur similaritas setiap titik data dengan *k* titik pusat klaster lalu meletakkan titik data itu ke dalam klaster yang terdekat dengannya. Oleh karena itu, kualitas algoritma K-Means sangat bergantung pada pemilihan k titik pusat awal klaster. Jika pemilihan tersebut tepat, maka algoritma *k*-Means akan berhasil mengelompokkan data secara baik. Sebaliknya, algoritma *k*-Means memiliki kecenderungan besar untuk terjebak dalam *local minima* apabila k titik pusat awal klaster yang dipilih tidak tepat.

Telah banyak penelitian dilakukan untuk mengembangkan dan memperbaiki kualitas kinerja *k*-Means terutama dalam pemilihan *k* titik pusat awal klaster. Beberapa penelitian menerapkan teori penambahan (*incremental*), yaitu mencari titik pusat sekaligus menambah jumlah klaster secara bertahap. Contoh metode-metode yang menerapkan cara demikian adalah *bisecting k-means* (Tan, 2006), *incremental k-means* (Pham, 2004), dan *PG-Means* (Feng, 2006). Adapula penelitian-penelitian yang memadukan metode optimasi, seperti algoritma genetika (*genetic algorithm)* (Martini, 2006) dan penguatan tersimulasi (*simulated annealing*) (Mualik, 2001) dalam tahap penentuan titik pusat awal klaster.

Penelitian untuk pengembangan dan perbaikan metode *k*-Means tidak hanya pada penentuan titik pusat awal klaster saja. Banyak pula usaha-usaha

untuk memperbaiki kecepatan komputasi dan efisiensi algoritmanya. Tahun 1998, Alsabti dkk melakukan penelitian dengan memasukkan set data ke dalam *kd-tree* untuk mengurangi waktu komputasi dan waktu akses ke database. Metode ini kemudian diperbaiki oleh Kanugo dkk (Kanugo, 2002). Adapula yang menambahkan sebuah variabel untuk menampung jarak setiap objek dengan titik pusat klaster dalam setiap iterasi (Fahim, 2006).

Hung, dkk (Hung, 2005) melakukan terobosan dalam efisiensi dan kecepatan komputasi *k*-Means, yaitu mereduksi data hingga menjadi beberapa titik representasi, disebut CUB (*center of unit block*). Dengan menerapkan algoritma *k*-Means, CUB-CUB itulah yang digunakan untuk mencari titik pusat awal klaster. Karena tidak memakai keseluruhan data dalam proses pencarian titik pusat, metode ini dapat melakukan mengurangi kompleksitas waktu hingga menjadi $O((n/u)\alpha k'd) + O((u-\alpha)kd)$ per iterasi, dimana algoritma dasar *k*-Means memiliki kompleksitas $O(n*k*d)$ per iterasi.

Meski efisiensi algoritma tersebut telah dibuktikan oleh Hung dkk namun dicurigai adanya kelemahan dalam pembentukan titik representasinya. Posisi titik representasi dalam algoritma tersebut merupakan nilai rerata dari sejumlah titik yang sebelumnya dipilah-pilah dalam unit-unit blok. Pembuatan unit blok itu sendiri dilakukan dengan membagi jarak rentang (*range*) nilai per dimensi dengan sebuah konstanta. Hal ini diduga akan mengakibatkan hilangnya korelasi antar dimensi dalam set data.

Selain itu, isi dari masing-masing blok unit sangat bevariasi. Sebuah CUB dapat mewakili banyak titik data, dan CUB lain mungkin hanya mewakili sebuah titik data. Demikian juga sebuah CUB dapat mewakili titik-titik data yang seharusnya memang anggota sebuah klaster, dan CUB lain mewakili titik data yang ternyata adalah desau (*noise)*. Setiap CUB mempunyai peluang yang sama untuk menjadi titik pusat awal klaster karena langkah awal fase penentuan titik pusat awal klaster pada algoritma Hung adalah memilih secara sembarang k CUB menjadi titik pusat awal klaster sebelum kemudian melakukan perhitungan dengan algoritma *k*-Means. Pemilihan secara random tersebut sangat bergantung pada peruntungan, karenanya hasil akhir yang dicapai tidak selalu optimal

Di sisi lain, terdapat metode klasterisasi yang berbasis pada kepadatan, salah satunya adalah DBSCAN. Metode ini menemukan kumpulan data yang berkepadatan tinggi, yang terpisah satu sama lain oleh kumpulan data yang berkepadatan rendah (Tan, 2006), dan bukan mencari jarak antar dimensi data, sehingga korelasi antar dimensi tetap terjaga.

Penelitian ini melakukan pendekatan berbasis kepadatan untuk membentuk titik representasi seperti pada algoritma Hung dkk. Metode yang digunakan berlandas pada pemikiran bahwa 1) sebuah titik pusat klaster pasti berada di tengah klaster (itu pula sebabnya algoritma k-means selalu menggeser posisi titik pusat ke tengah titik-titik dalam klaster), dan 2) semakin ke pusat, kepadatan data semakin tinggi. Oleh karena itu, titik representasi akan dibentuk dari kumpulan titik data yang berkepadatan tinggi saja. Praduga awal dari metode ini adalah titik representasi yang terbentuk relatif lebih sedikit daripada yang terbentuk dari metode Hung dkk, namun sekaligus lebih representatif.

1.2 Perumusan Masalah

Permasalahan utama dari penelitian ini adalah bagaimana membentuk titik representasi berbasis kepadatan.

Titik representasi digunakan untuk mempercepat iterasi dalam proses penentuan titik pusat awal klaster. Oleh karena itu sebuah titik representasi akan mewakili sekumpulan titik data. Jika pada algoritma Hung, dkk titik-titik representasinya tersebar hampir merata dalam ruang data, maka penelitian ini ingin menghasilkan titik-titik representasi yang berada di tengah/dalam klaster sebab titik pusat klaster umumnya berada di tengah klaster. Dengan kata lain, diharapkan titik-titik representasi tersebut akan mewakili titik-titik yang berada di tengah/dalam klaster saja.

Bertolak dari pemikiran tersebut, permasalahan kemudian bergeser menjadi: 1) bagaimana menemukan titik-titik data di tengah klaster, dan 2) bagaimana menentukan kumpulan titik data yang diwakili sebuah titik representasi.

1.2.1 Bagaimana Menemukan Titik-Titik Data di Tengah Klaster

Secara kasat mata, apabila melihat visualisasi sebuah set data dalam dua dimensi, sebuah klaster dapat diindikasikan dengan adanya kerumunan-kerumunan titik data pada set data. Sehingga, secara kasat mata pula, akan sangat mudah untuk menunjuk titik-titik mana saja yang berada di tengah klaster. Penentuan atau penunjukkan itu menjadi rumit jika ternyata set data itu memiliki lebih dari dua dimensi (atribut) dan algoritma hanya dapat 'melihat' posisi titik data dari nilai setiap dimensinya.

Namun, kerumunan titik data itu (yang nantinya adalah sebuah klaser) pada umumnya cenderung memadat di tengah. Jarak antar titik data di tengah/pusat kerumunan biasanya lebih pendek sedangkan jumlah titik datanya jauh lebih banyak daripada titik-titik data di tepi kerumunan. Sehingga apakah sebuah titik data berada di tengah klaster dapat ditentukan dengan menghitung apakah titik data tersebut berada di daerah yang padat atau tidak.

1.2.2 Bagaimana Menentukan Kumpulan Titik Data yang Diwakili Sebuah Titik Representasi

Setelah titik-titik data yang berada di tengah klaster ditemukan, maka titik-titik data itu haruslah dikelompok-kelompokkan dalam beberapa grup agar kemudian dapat diwakili oleh sebuah titik representasi. Idealnya, sebuah titik data dikelompokkan bersama titik-titik data yang dekat atau bertetangga dengannya. Jika hanya mengindahkan titik yang saling bertetangga langsung saja, ditakutkan akan terdapat titik data yang tidak layak masuk dalam grup manapun. Oleh karena itu, pengelompokkan tidak hanya berdasarkan titik tetangga melainkan rantai titik tetangga, yaitu tetangga dari tetangga sebuah titik data.

1.2.3 Bagaimana Menemukan Titik-Titik Data di Tengah Klaster Tanpa Menggunakan Parameter Radius Kepadatan

Pemecahan masalah yang dijelaskan pada sub subbab 1.2.1 nantinya memerlukan dua buah nilai konstanta, yaitu batas minimum kepadatan untuk menentukan apakah daerah di sekitar titik data termasuk padat, dan radius kepadatan yang menentukan rentang daerah yang dihitung. Keduanya merupakan

parameter yang harus diketahui dari awal. Dengan kata lain, pengguna algoritma yang harus memberikannya.

Tambahan dua parameter ini menimbulkan permasalahan. Sebab untuk mendapatkan hasil yang optimal, nilai kedua parameter tersebut akan berubah-ubah sesuai dengan set data yang digunakan. Permasalahan ini diperburuk dengan kondisi pengguna algoritma yang, pada kebanyakan kasus, tidak mengetahui nilai optimal parameter-parameter tersebut. Akibatnya, kemungkinan algoritma akan gagal (berkualitas buruk) karena kesalahan pemberian nilai parameter sangat besar. Di lain pihak, penentuan nilai secara otomatis untuk kedua parameter itu sekaligus merupakan hal yang sangat rumit karena keduanya berkaitan satu sama lain dengan erat.

Hal yang dapat dilakukan untuk mengurangi kemungkinan gagal itu adalah dengan menentukan secara otomatis salah satu nilai konstanta itu. Dalam penelitian ini, parameter yang akan dihilangkan adalah radius kepadatan.

Penghapusan parameter radius kepadatan mengubah langkah-langkah solusi permasalahan sub subbab 1.3.1, namun tidak secara prinsip Untuk menemukan titik data di tengah klaster tidak lagi dengan melihat kepadatan sebuah titik data, tapi dengan melihat apakah sebuah titik data termasuk dalam daerah kepadatan titik tetangga terdekatnya.

1.3 Tujuan dan Manfaat

Tujuan utama dari penelitian ini adalah memperbaiki kualitas klasterisasi *k*-Means melalui penentuan titik pusat awal klaster menggunakan titik representasi berbasis kepadatan.

BAB 2
KAJIAN PUSTAKA

Bab ini membahas beberapa kajian pustaka yang menjadi landasan teori dan pangkal keberangkatan penelitian ini. Kajian tersebut antara lain adalah mengenai algoritma dasar k-Means, algoritma k-Means dengan partisi sederhana, dan algoritma klasterisasi berbasis kepadatan.

2.1 Algoritma Dasar K-Means

Algoritma klasterisasi *k*-Means bekerja dengan prototype, yaitu titik pusat klaster, dan menempatkan titik-titik dalam set data ke dalam salah satu dari k kelompok berdasarkan kedekatan titik data itu dengan titik pusat klaster. Sebagai langkah awal, algoritma ini mengambil sembarang *k* titik dalam set data sebagai titik pusat awal klaster, dimana k adalah jumlah klaster. Lalu algoritma ini menghitung jarak Euclidean titik data x_i dengan k titik pusat klaster dan menunjuk klaster terdekat (yang diwakili oleh titik pusat klaster) sebagai klaster dari titik data tersebut. Letak titik pusat klaster kemudian diperbarui hingga berada di tengah-tengah kumpulan titik data yang merupakan anggota klasternya. Langkah penempatan titik data dan pembaruan letak titik pusat ini diulang hingga letak titik pusat tidak berubah.

Algoritma *k*-Means secara formal dapat ditulis sebagai berikut:

a. Pilih sembarang k titik dari n set data sebagai titik pusat awal $\{v_1, v_2, \ldots, v_k\}$
b. Tempatkan setiap titik data x_i pada klaster terdekat, yaitu yang titik pusat klaster berjarak Euclidean *l* terdekat dengan x_i.

$$l = \sqrt{\sum_{m=1}^{d} (xi - vj)^2} \text{ , untuk i = 1,..., n dan j = 1,... , k} \qquad (2.1)$$

c. Re-lokasi titik pusat klaster v_j agar berada di pusat klaster,

$$vj = \frac{\sum xi}{|xi|} \tag{2.2}$$

Hitung juga fungsi kesalahan E

$$E = \sum_{j=1}^{k} \sum_{xi \in Cj} \|x_i - v_j\|^2 \tag{2.3}$$

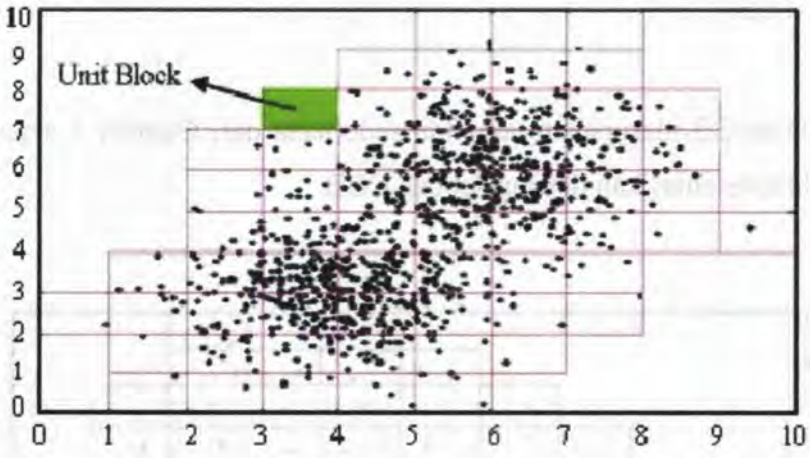
d. Ulangi langkah 2 hingga posisi v_j tidak berubah atau nilai fungsi kesalahan E berada di bawah batas (*threshold*).

2.2 Algoritma K-Means dengan Partisi Sederhana

Algoritma ini adalah salah satu dari sekian banyak pengembangan algoritma dasar *k*-Means yang bertujuan untuk mempercepat waktu komputasi, yaitu dengan menggunakan partisi sederhana dalam mencari titik pusat klaster.

2.2.1 Pembagian Set Data ke dalam Blok – Blok Unit

Fase pertama dari algoritma ini adalah mempartisi set data kedalam blok-blok unit (*unit blocks*), yang untuk seterusnya disebut UB. Pembuatan UB dilakukan dengan sebelumnya menghitung nilai maksimum dan minimum data di setiap dimensi dan menjadikan nilai-nilai tersebut sebagai batas ruang data. Untuk data dua dimensi, ruang data itu akan berbentuk persegi panjang. Kemudian, masih dalam tiap dimensi, dilakukan n pemotongan terhadap ruang data sehingga menghasilkan 2^n blok-blok yang berukuran sama besar. Jumlah pemotongan ini merupakan sebuah konstanta yang mana dalam eksperimennya, Hung, dkk menemukan 11 sebagai nilai yang memberikan hasil optimal. Setelah UB terbentuk, maka setiap titik data ditempatkan dalam UB yang sesuai. Gambar 2.1 memperlihatkan hasil dari proses pembentukan UB di atas.



Gambar 2.1 Set Data yang Terbagi dalam Segmen-segmen Blok Unit.

2.2.2 Pembentukan Titik Representasi untuk Tiap Unit Blok

Langkah selanjutnya adalah membuat titik representasi yang akan mewakili titik-titik data di dalam setiap UB. Langkah ini berfungsi untuk menyederhanakan tampilan set data dan, tentu saja, akan mempercepat waktu komputasi. Pembuatan titik representasi sendiri dilakukan bersamaan dengan penempatan titik-titik data kedalam UB. Titik representasi ini disebut dengan *center of unit block* (CUB) karena merupakan titik rerata (*average point*) dari titik-titik data di dalam sebuah unit blok. Apabila CUB_a adalah sebuah titik representasi dari UB ke-a, maka notasi formal CUB_a adalah :

$$CUB_a = \frac{LSUB_a}{WUB_a}; a \in \{1,2,...,u\} \tag{2.4}$$

dengan

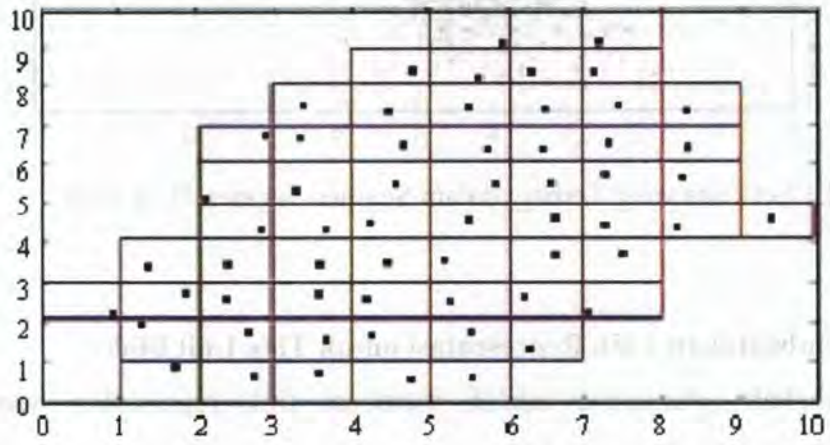
- $LSUB_a$ sebagai penjumlahan linier per dimensi untuk setiap titik data di UB ke-a, atau

$$LSUB_a = \sum_{x_i \in UB_a} x_i \tag{2.5}$$

- WUBa sebagai berat UB ke-a yang diperoleh dari totol jumlah titik data di UB ke –a

$$WUB_a = |UB_a| \tag{2.6}$$

Gambar 2.2 merupakan tampilan sederhana dari Gambar 1 setelah setiap unit blok direpresentasikan dengan sebuah CUB.



Gambar 2.2. Tampilan Sederhana Set Data yang Telah Tereduksi.

2.2.3 Penentuan Titik Pusat Awal Klaster

Algoritma ini kemudian beralih ke fase berikutnya, yaitu mencari titik pusat klaster dengan menerapkan algoritma *k*-Means pada data tereduksi tersebut (daa yang direpresentasikan oleh CUB). Langkah-langkah formalnya adalah sebagai berikut :

a. Pilih k sembarang CUB sebagai inisialisasi titik pusat klaster $\{v_1, v_2, .., v_j\}$

b. Hitung jarak Euclidean *l* masing-masing CUB dan titik pusat dalam setiap dimensi.

$$l = \sqrt{\sum_{m=1}^{d} (CUB_a - v_j)^2} \tag{2.7}$$

Setiap CUB akan menjadi anggota salah satu dari k klaster yang berjarak Euclidean terdekat dengannya.

c. Relokasi setiap titik pusat klaster v_j dengan persamaan 2.8 dan hitung fungsi kesalahan E dengan persamaan 2.9 berikut:

$$v_j = \frac{\sum_{CUB_a \in C_j} LSUB_a}{\sum_{CUB_a \in C_j} WUB_a} \tag{2.8}$$

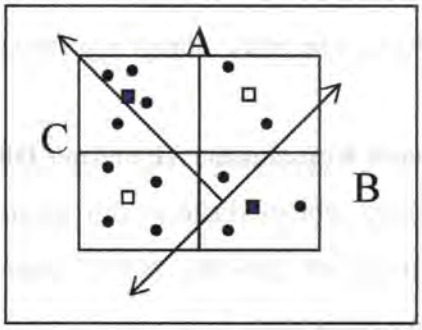
$$E = \sum_{j=1}^{k} \left(\sum_{CUB_a \in C_j} WUB_a \left\| CUB_a - v_j \right\|^2 \right) \tag{2.9}$$

d. Ulangi proses pencarian titik pusat hingga fungsi kesalahan E tidak berubah secara signifikan.

2.2.4 Penempatan Titik Data dalam Klaster

Setelah k titik pusat awal berhasil ditemukan, maka fase selanjutnya adalah menempatkan semua titik data ke dalam klaster-klaster yang berkesesuaian. Namun perlu diingat bahwa titik pusat yang dihasilkan dari set data yang tereduksi ini belum tentu adalah titik pusat klaster yang sebenarnya, sebab CUB yang terletak di batas klaster tidak dapat mewakili semua titik data di UB pada fase penempatan akhir ini.

Karena UB merupakan hasil segmentasi per dimensi, maka sangat mungkin bila sebuah UB terletak diantara dua klaster atau disebut juga berada pada batas klaster. Situasi ini tergambarkan pada Gambar 2.3. Keberadaan CUB-CUB berwarna biru tidak dapat mewakili semua titik data dalam unit blok masing-masing, sebab dalam set data yang sebenarnya, titik-titik data tersebut tidak berada dalam klaster yang sama.



Gambar 2.3. Unit Blok yang Berada pada Batas Klaster dan yang Tidak Berada pada Batas Klaster.

Secara matematis, sebuah UB_a dikatakan berada pada batas klaster jika selisih jarak CUBa dengan titik pusat klaster terdekat pertama v_j dan jarak CUB_a dengan titik pusat klaster terdekat kedua v_j' lebih kecil daripada panjang diagonal UB_a.

Oleh karena itu untuk penentuan akhir klaster, algoritma ini menggunakan set data tereduksi (CUB) untuk unit blok yang tidak berada pada batas dan set data asli untuk unit blok yang berada pada batas klater dengan langkah-langkah sebagai berikut :

a. Jika UB berada pada batas klaster, atau dengan kata lain $(\|CUB_a - v_j\| < \|CUB_a - v_j'\|) < L_d$ dengan L_d adalah panjang diagonal UB_a, maka cari klaster terdekat untuk setiap titik data x_i dari set data asli, yaitu dengan menghitung jarak x_i dengan titik pusat klaster terdekat pertama v_j dan jark xi dengan titik pusat klaster terdekat kedua v_j'.
b. Jika UB tidak berada pada batas klaster, maka pergunakan prosedur sebelumnya untuk menhitung titik pusat terdekat.
c. Relokasi titik pusat klaster v_j dan htiung fungsi kesalahan E

$$vj = \frac{\sum_{(CUB_a \succ BUNB) \in C_j} LSUB_a + \sum_{(Xi \succ BUB) \in C_j} x_i}{\sum_{(CUB_a \succ BUNB) \in C_j} WUB_a + |Xi \succ BUB) \in C_j|} \tag{2.10}$$

$$E = \sum_{j=1}^{k} \left(\sum_{(CUB_a \succ BUNB) \in C_j} WUBa \|CUBa - vj\|^2 + \sum_{(x_a \succ BUB) \in C_j} \|xi - vj\|^2 \right) \tag{2.11}$$

d. Jika perubahan fungsi kesalahan E berada di bawah batas yang ditentukan, maka akhiri algoritma. Jika tidak, ulangi algoritma.

2.3 Klasterisasi Berbasis Kepadatan: Algoritma DBSCAN

Sesuai dengan definisi sebuah klaster, yaitu sekumpulan objek (data) yang memiliki kesamaan dalam level tertentu, maka klasterisasi dapat dilakukan melalui pendekatan berbasis kepadatan.

> Alasan sebuah klaster dapat dikenali adalah karena di dalam klaster terdapat kepadatan titik-titik yang sangat tinggi bila dibandingkan dengan kepadatan di luar klaster. (Ester 1996)

DBSCAN adalah algoritma klasterisasi berbasis kepadatan yang menggunakan pendekatan titik pusat. Pada DBSCAN, kepadatan setiap titik dihitung dengan mencari jumlah titik ketetanggaannya dalam radius tertentu, yang disebut *Eps*. Fungsi jarak yang digunakan untuk menghitung ketetanggan dapat beragam dan disesuaikan dengan kebutuhan aplikasi. Sebuah batas minimum jumlah titik ketetanggaan, disebut *MinPts*, juga ditetapkan dari awal.

Batas titik ketetanggaan minimum, dinotasikan *MinPts*, adalah jumlah titik ketetanggaan (termasuk dirinya) yang harus dimiliki sebuah titik agar dapat diklasifikasikan menjadi titik inti.

Titik inti (*core point*) adalah titik yang memiliki titik ketetanggaan paling tidak sebanyak *MinPts*. Sebuah titik inti dapat dipastikan menjadi anggota sebuah klaster dan berada di dalam / tengah klaster.

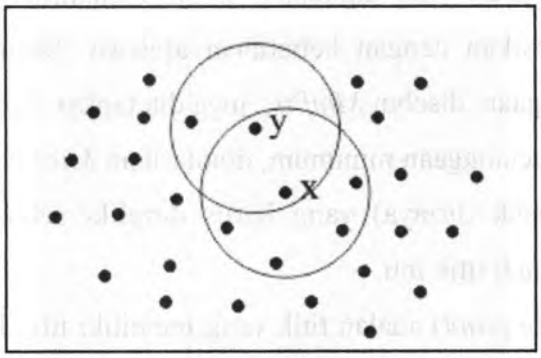
Titik tepi (*border point*) adalah titik yang memiliki titik ketetanggaan kurang dari *MinPts*. Sebuah titik tepi dapat menjadi anggota sebuah klaster atau menjadi sebuah desau. Sebuah titik tepi y adalah anggota sebuah klaster C bila :

- Titik tepi y berada dalam ketetanggaan titik inti x yang merupakan anggota dari klaster C, disebut jangkauan kepadatan langsung (*directly density-reachable*). Kondisi ini ditunjukkan Gambar 2.4.
- Terdapat rantai titik p_1, p_2, ..., p_n antara titik tepi y dengan titik inti x ($x \in C$), $p_1 = y$ dan $p_n = q$ sedemikian rupa sehingga p_{i+1} adalah jangkauan kepadatan langsung dari p_i .Kondisi demikian, yang ditunjukkan oleh Gambar 2.5, disebut jangkauan kepadatan (*density reachable*).
- Terdapat sebuah titik inti x yang adalah anggota klaster C, dan sebuah titik tepi lain z sedemikian rupa hingga y dan z sama-sama merupakan jangkauan kepadatan dari x. Hubungan kepadatan ini (*density-connected*) ditunjukkan oleh Gambar 2.6.

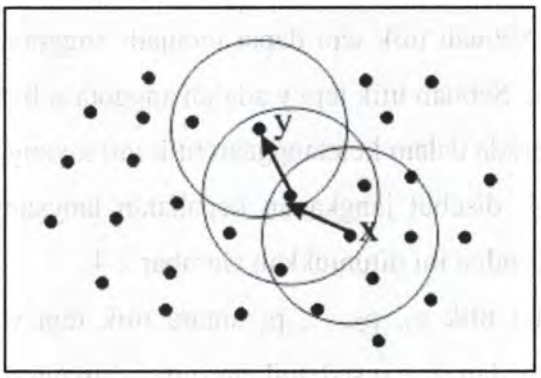
Berdasarkan definisi 5 dari penelitian Ester, dkk (Ester 1996), sebuah klaster C adalah sebuah subset tidak kosong dari set data D yang setiap titiknya merupakan jangkauan kepadatan dari sebuah titik inti, atau paling tidak memiliki hubungan kepadatan satu dengan yang lain.

Sebuah desau adalah titik tepi yang tidak termasuk ke dalam klaster manapun. Dengan kata lain, desau adalah titik tepi yang tidak memiliki jangkauan

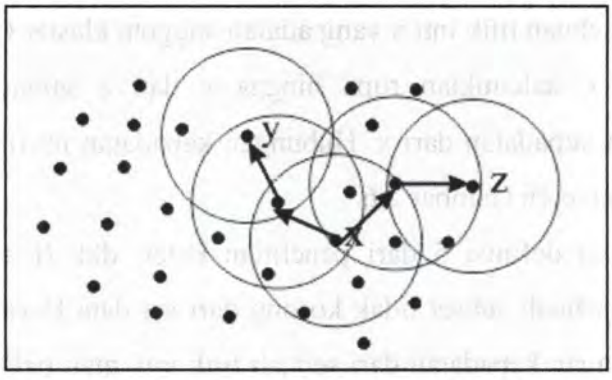
kepadatan dengan titik inti manapun dan juga hubungan kepadatan dengan titik tepi yang lain.



Gambar 2.4. Titik Tepi y Merupakan Jangkauan Kepadatan Langsung Terhadap Titik Inti x



Gambar 2.5. Titik Tepi y Merupakan Jangkauan Kepadatan dari Titik Inti x



Gambar 2.6. Titik Tepi y dan Titik Tepi z Mempunyai Hubungan Kepadatan Melalui Titik Inti x

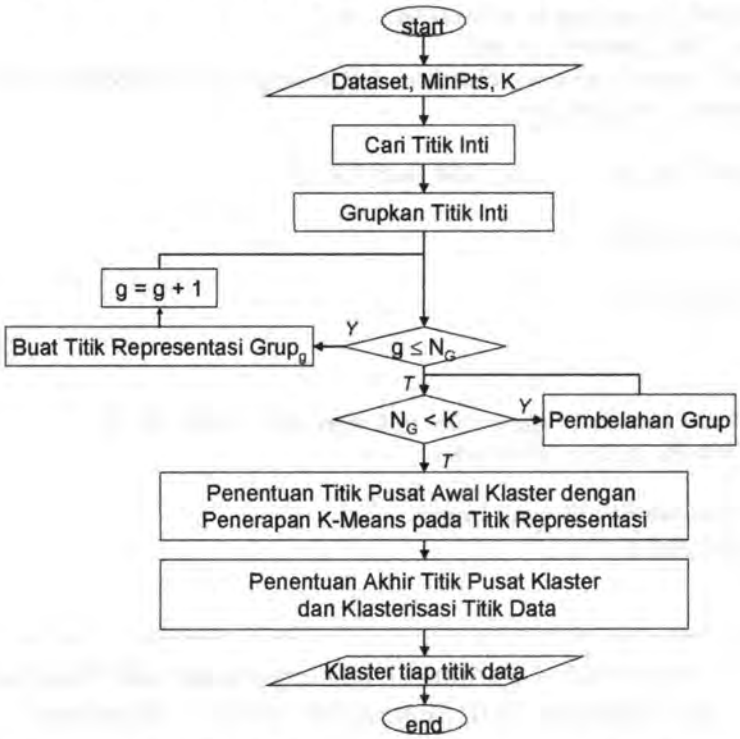
BAB 3

DESAIN ALGORITMA

Penelitian ini merupakan pengembangan dari algoritma *k*-Means dengan partisi sederhana yang dikembangkan Hung, dkk. (Hung, 2005). Sesuai dengan tujuan penelitian, bagian dari algoritma terdahulu yang hendak dikembangkan atau diperbaiki adalah bagian pembuatan titik representasi. Dalam penelitian ini titik representasi dibuat dari pengelompokkan titik-titik inti. Perubahan lain yang dilakukan adalah inisialisasi titik pusat awal klaster yang tidak lagi dilakukan secara sembarang (*random*), melainkan dengan mengambil titik-titik yang paling banyak merepresentasikan titik inti

3.1 Algoritma Utama

Secara umum, algoritma yang dikembangkan dalam penelitian ini memiliki aliran proses/modul seperti pada Gambar 3.1. berikut ini :



Gambar 3.1. Diagram Alir Modul-Modul pada Algoritma Penelitian

Algoritma dalam penelitian ini terdiri dari beberapa modul (tampak pada Algoritma 3.1). Modul yang pertama adalah modul pencarian titik inti yang berbasis kepadatan data pada set data. Modul ini dijelaskan pada sub-bab 3.2. Modul kedua berfungsi untuk membentuk grup-grup titik inti dengan cara mengelompokkan titik-titik inti berdasarkan jarak antar titik inti, diterangkan dalam sub-bab 3.3, yang dilanjutkan dengan pembentukan titik representasi pada sub-ba 3.4. Jika jumlah grup yang terbentuk kurang dari jumlah perkiraan klaster, maka grup akan dibelah hingga berjumlah sama dengan jumlah perkiraan klaster. Sub-bab 3.6 menjelaskan tentang penerapan algoritma k-Means pada titik-titik representasi yang berfungsi untuk menentukan titik pusat awal klaster. Setelah titik pusat awal klaster ditentukan, maka dilakukan penentuan akhir titik pusat klaster dan klasterisasi seluruh titik-titik data, yang terangkum sub-bab 3.7.

```
ALGORITMA Utama (dtset, MinPts, K)
//Algoritma utama dari penelitian
//Input   : set data yang ingin diklaster, jumlah MinPts, jumlah perkiraan klaster
//Output : klaster tiap titik Cj dengan j antara 1 sampai dengan K
BEGIN
    //cari titik inti dan masukkan dalam koleksi Inti
    Inti = Cari_Titik_Inti(dtSet,MinPts)
    //NG adalah jumlah grup yang dihasilkan dari proses pengelompokkan titik inti
    NG = Grupkan_Titik_Inti(Inti)

    FOR setiap Grupg, g∈ {1,..,NG} DO Buat_TR(g)

    WHILE NG < K DO
    BEGIN
        Belah_Grup(NG)
        NG = NG + 1
    END

    //Inisiasi k titik pusat awal klaster dari titik representasi grup inti TRg
    Inisiasi_Titik_Pusat_Dari_TR (NG,K)

    //Perhitungan akhir k titik pusat klaster
    Cj =KMeans_Akhir
    Return Cj
END
```

Algoritma 3.1. *Pseudocode* Algoritma Utama Penentuan Titik Pusat Awal Klaster Menggunakan Titik Representasi Berbasis Kepadatan.

Kompleksitas waktu algoritma dalam penelitian ini adalah $O_{Total} = O_1 + O_2 + N_G*O_3 + (K-N_G)*O_4 + O_5 + O_6$, dengan O_1 sampai dengan O_6 adalah

kompleksitas waktu dari masing-masing modul. Modul pembentukan titik inti (sub-bab 3.4) akan dilakukan sebanyak jumlah Grup N_G sehingga membutuhkan waktu N_G*O_3, sedangkan Modul pembelahan grup (sub-bab 3-5) akan dilakukan sebanyak K-N_G kali.

3.2 Algoritma Pencarian Titik Inti pada Set Data

Pencarian titik inti di sini mengadaptasi algoritma klasterisasi berbasis kepadatan seperti yang telah dijelaskan pada subbab 2.3. Namun algoritma pencarian titik inti pada metoda ini memiliki beberapa perbedaan dengan algoritma klasterisasi berbasis kepadatan, yaitu : a) algoritma ini hanya mengindahkan titik inti dan titik tepi. Desau dianggap sebagai titik tepi. b) Tidak diperlukannya parameter radius kepadatan, *Eps*. Pengguna hanya perlu menentukan batas minimum kepadatan, *MinPts* saja. *Pseudocode* dari algoritma pencarian titik inti ini digambarkan pada Algoritma 3.2 berikut :

```
ALGORITMA Cari_Titik_Inti (dtset, MinPts)
//Algoritma pencarian titik inti dari dataset
//Input   : set data yang ingin diklaster, jumlah MinPts
//Output : koleksi titik inti, Inti, yang adalah variable global
BEGIN
    //Cari tetangga terdekat setiap titik
    FOR setiap x_i , i ∈ {1,...n} DO Nearest_Neighbor(x_i , MinPts-1)
    AvgEps = mean(Eps)
    //Penentuan titik inti
    FOR setiap x_i , i ∈ {1,...n} DO
        IF dmin[x_i] ≤ Eps[x_ci] and Eps[x_ci] ≤ AvgEps THEN
        BEGIN
            x_i ∈ Inti
            dependent[x_ci] = dependent[x_ci] + x_i
        END
        ELSE
        BEGIN
            x_i ∉ Inti
            N_T = N_T + 1
            IF dependent[x_i] tidak kosong THEN Ubah_Jadi_Tepi(x_i)
        END
    Return Inti
END
```

Algoritma 3.2. *Pseudocode* Algoritma Pencarian Titik Inti

```
ALGORITMA Nearest_Neighbor (y, t)
//Algoritma pencarian tetangga terdekat
//Input  : titik y yang dicari tetangga terdekatnya, jumlah tetangga terdekat t
//Output : list tetangga terdekat NN dari y yang adalah variabel global
BEGIN
    z = 1
    FOR setiap x_i , i ∈ {1,...n), x_i ≠ y DO
    BEGIN
        FOR setiap dimensi d dari x_i DO
            d(y, x_i) = √(Σ(y − x_i)^2)
        IF z ≤ t THEN
        BEGIN
            x_i ∈ NN[y]
            IF d(y,x_i) < dmin[y] THEN
            BEGIN
                x_ci = x_i   // x_ci adalah titik terdekat dari titik y
                dmin[y] = d(y,x_i)
            END
            IF d(y,x_i) > Eps[y] THEN
            BEGIN
                x_max = x_i // x_max adalah tetangga terdekat ke-n dari titik y
                Eps[y] = d(y,x_i) //Eps[y] adalah jarak x_max ke y
            END
            z = z +1
        END
        ELSE IF d(y,x_i) < Eps[y] THEN //jika list NN penuh tapi jarak lebih kecil dr Eps
        BEGIN
            Gantikan tempat x_max di NN [y] dengan x_i
            urutkan NN [y] berdasarkan jarak
            x_ci = x pertama di NN[y]
            x_max = x terakhir di NN[y]
            dmin[y] = d(y,x_ci)
            Eps[y] = d(y,x_max)
        END
    END
    Return NN[y]
END
```

Algoritma 3.3. *Pseudocode* Metode Ketetanggan Terdekat

Proses pencarian titik diawali dengan mencari sejumlah *MinPts - 1* titik terdekat dari masing-masing titik data dengan menggunakan algoritma Ketetanggan Terdekat (*Nearest-Neighbor*) yang *pseudocode*-nya seperti pada Algoritma 3.3. Kemudian algoritma akan menyimpan jarak ketetanggaan terdekat setiap titik dengan notasi *dmin*, dan jarak ketetanggaan terjauh dari setiap titik sebagai *Eps*.

Dmin adalah jarak sebuah titik dengan tetangga terdekat pertamanya.

$$dmin(x_i) = \min d(x_i, x_i') \text{ dimana } x_i' \in NN(x_i) \quad (3.1)$$

$x_{ci} = x_i'$ dimana $d(x_i,x_i') = dmin(x_i)$ (3.2)

Eps adalah jarak sebuah titik dengan tetangga terdekat ke-t dengan t = MinPts - 1. dengan kata lain, *Eps* adalah jarak terjauh sebuah titik dengan tetangga terdekatnya.

$Eps(x_i) = \max d(x_i,x_i')$ dimana $x_i' \in NN(x_i)$ (3.3)

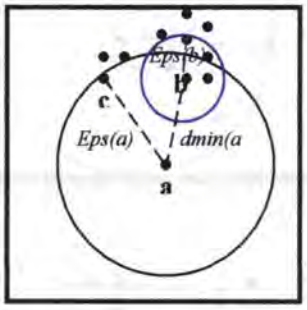
Dmin dan *Eps* tersebut digunakan pada langkah selanjutnya, yaitu penentuan titik inti. Sebuah titik x_i dinyatakan sebagai titik inti apabila *dmin* dari x_i lebih kecil / sama dengan *Eps* dari x_{ci}, titik tetangga terdekat pertama x_i.

$$x_i \begin{cases} \in inti, \text{ if } dmin(x_i) \le Eps(x_{ci}) \\ \notin inti, \text{ else} \end{cases} \quad (3.4)$$

Secara visual, pernyataan di atas dapat dijelaskan seperti Gambar 3.2. Apabila terdapat set data dengan sebaran titik data demikian dan nilai *MinPts* yang diberikan pengguna adalah 4, maka kepadatan dari titik a digambarkan oleh lingkaran hitam, dengan :

- Nilai *Eps* titik a adalah jarak ketetanggan titik a dengan titik c,
- Nilai *dmin* titik a adalah jarak ketetanggaan titik a dengan titik b.

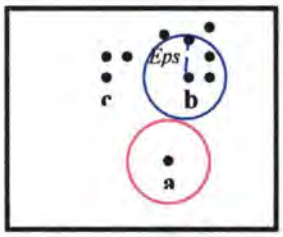
Sedangkan kepadatan titik b, sebagai titik ketetanggaan terdekat dari titik a, digambarkan oleh lingkaran biru.



Gambar 3.2. Penggambaran Proses Penentuan Titik Inti dengan *MinPts* = 4.

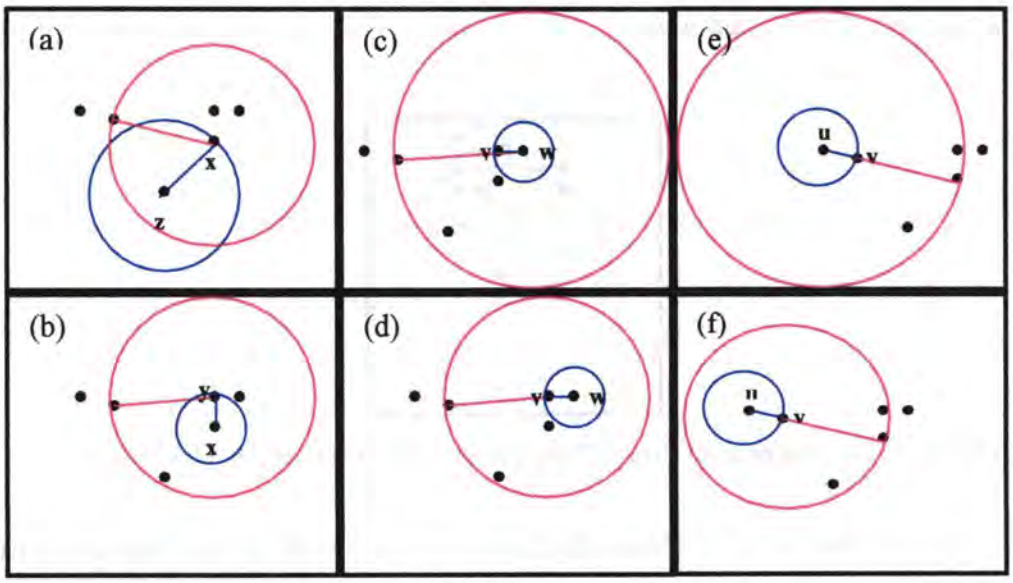
Pada Gambar 3.2, terlihat jelas bahwa titik a berada di luar lingkaran biru, atau dengan kata lain titik a tidak termasuk dalam kepadatan titik b, sehingga titik a adalah titik tepi (bukan titik inti). Analisa tersebut dapat dijelaskan dalam sudut

pandang metode berbasis kerapatan sebagai berikut: Jika parameter *MinPts* adalah 4 dan *Eps* adalah *Eps(b)* (di sini, *Eps(b)* adalah sebuah konstanta yang diberikan pengguna, bukan lagi dari perhitungan jarak terjauh dari titik b), maka bagi titik b, titik a tidak termasuk dalam kepadatannya. Sedangkan kepadatan titik a adalah 1 karena tidak ada titik yang berada pada jarak *Eps(b)* darinya kecuali dirinya sendiri. Hal ini berarti titik a adalah titik tepi, karena kepadatannya kurang dari *MinPts*. Visualisasi dari analisa di atas ditunjukkan oleh Gambar 3.3.



Gambar 3.3. Penentuan Titik Inti dengan Metode Berbasis Kepadatan dengan *MinPts* = 4 dan Eps diasumsikan adalah Eps(b)

Dalam tahap pembuatan metode ini, ditemukan fakta bahwa sangat jarang terdapat titik yang benar-benar terisolir. Dengan kata lain, setiap titik hampir selalu berada dalam radius kepadatan titik lain. Ilustrasi dari fakta ini tampak pada Gambar 3.4.



Gambar 3.4. Ilustrasi Data yang Tidak Memiliki Titik Tepi

Pada Gambar 3.4(a), titik z letaknya cukup berjauhan dengan titik-titik lain. Dengan *MinPts* = 5, titik tetangga terdekat titik z adalah titik x, sehingga dmin(z) adalah garis biru dengan radius lingkaran biru, dan Eps(x) adalah garis merah dengan radius lingkaran merah. Dari gambar tersebut diketahui bahwa titik z adalah titik inti karena berada dalam radius kepadatan titik x. Sedangkan titik x adalah titik inti dengan merujuk titik y (Gambar 3.4(b)). Demikian pula titik-titik lainnya adalah titik inti dengan merujuk satu sama lain, seperti tampak pada Gambar 3.4(c) sampai dengan 3.4(f).

Jika semua titik adalah titik inti, maka metode ini tidak dapat bekerja secara efektif. Karena itu berarti titik representasi yang dibentuk akan mewakili seluruh data, bukan hanya titik inti, sehingga titik pusat awal klaster kemungkinan akan berada di luar daerah tengah klaster. Untuk mengatasi hal itu, algoritma ini menambahkan sebuah kondisi yang harus dipenuhi agar sebuah titik dapat menjadi titik inti yaitu apabila Eps-nya tidak lebih besar dari nilai rerata Eps seluruh titik. Hal ini dilakukan dengan asumsi bahwa antara satu klaster dengan klaster yang lain mempunyai kepadatan yang tidak jauh berbeda. Sehingga Rumus 3.4 berubah menjadi:

$$x_i \begin{cases} \in inti, \text{ if } dmin(x_i) \le Eps(x_{ci}) \text{ AND } Eps(xci) \le AvgEps \\ \notin inti, \text{ else} \end{cases} \quad (3.5)$$

Jika titik x_i adalah titik inti, maka algoritma menyimpannya sebagai titik yang merujuk (dependent) pada titik x_{ci}. Bila ternyata selanjutnya didapati bahwa titik x_{ci} adalah titik tepi (bukan titik inti), maka titik x_i dan titik-titik lain yang merujuk pada x_{ci} akan diubah menjadi titik tepi. Titik-titik yang merujuk pada x_i pun diubah menjadi titik tepi. Sehingga algoritma pengubahan ke titik tepi ini dilakukan secara rekursif, seperti pada *pseudocode* Algoritma 3.4.

Kompleksitas waktu yang diperlukan oleh modul ini dapat diperoleh dengan perhitungan sebagai berikut :

a. Algoritma Nearest_Neighbor melakukan perulangan sebanyak (N-1), yang mana di dalamnya melakukan perhitungan jarak antar titik yang membutuhkan waktu (N-1)*d. Hal lain yang memerlukan waktu yang signifikan adalah pengurutan ulang titik-titik tetangga terdekat y setiap kali jarak sebuah titik x_i

lebih kecil dari jarak tetangga terdekat ke-t. Proses ini membutuhkan waktu t^2. Dengan t = *MinPts* -1, maka waktu dari penukaran tersebut adalah ((N-1)-(*MinPts*-1))*(*MinPts*-1)^2. Kompleksitas waktu dari algoritma Nearest_Neighbor adalah O(N-1)*(d+(*MinPts*-1)^2)– (*MinPts*-1)^3).

b. Algoritma Ubah_Jadi_Tepi yang dipanggil secara rekursive sebanyak jumlah titik tepi membutuhkan waktu O(N_I).

c. Algoritma pencarian titik inti memanggil algoritma Nearest_neighbor dan mengeset sebuah titik menjadi titik inti sebanyak jumlah titik (N) kali. Sehingga total kompleksitas waktu yang dibutuhkan adalah:
O(N*((N-1)*(d+(*MinPts*-1)^2)– (*MinPts*-1)^3)) + N_I)

```
ALGORITMA Ubah_Jadi_Tepi (y)
//Algoritma pengubahan titik-titik yang merujuk ke titik y menjadi titik tepi
//Input   : titik y yang merupakan titik rujukan
BEGIN
    //dependent[y] adalah list titik yang merujuk ke titik y
    FOR setiap x_d ∈ dependent[y] , x_d ∈ Inti DO
    BEGIN
        x_d ∉ Inti
        N_T = N_T + 1
        IF dependent[x_d] tidak kosong THEN Ubah_Jadi_Tepi(x_d)
    END
END
```

Algoritma 3.4. *Pseudocode* Perubahan Ke Titik Tepi

3.3 Algoritma Pengelompokkan Titik Inti

Setelah ditemukan, maka masing-masing titik inti akan dikelompokkan satu sama lain sehingga membentuk beberapa grup titik inti. Karena mulai dari sub-bab ini pembahasan lebih terarah pada titik inti, bukan lagi titik data secara keseluruhan, maka demi penyederhanaan notasi untuk selanjutnya notasi x adalah notasi titik inti, kecuali disebutkan lain. Sebuah titik inti x_i' akan berada dalam grup yang sama dengan titik inti x_i bila:

- titik inti x_i' adalah jangkauan kepadatan langsung dari titik inti x_i, atau
- titik inti x_i' adalah jangkauan kepadatan dari titik inti x_i.

Konsep ini berdasarkan pada pernyataan Ester, dkk (Ester, 1996), yaitu : "jangkauan kepadatan langsung (*directly density-reachable*) bersifat simetris

untuk pasangan titik inti" dan "walaupun secara umum tidak simetris, adalah jelas bahwa jangkauan kepadatan (*density-reachable*) bersifat simetris untuk titik-titik inti)".

Berdasarkan konsep tersebut, maka algoritma ini mulanya akan mencari titik-titik inti yang merupakan jangkauan kepadatan langsung dari titik inti x_i, sebutlah sebagai x_m. Karena radius kepadatan *Eps* adalah jarak sebuah titik dengan tetangga terdekat ke-t, dimana t = *MinPts* – 1, maka x_m dapat dicari dalam kumpulan titik tetangga terdekat dari x_i. Jika x_m belum mempunyai grup, maka masukkan x_m dalam grup yang sama dengan x_i.

$x_m \in Grup_g$ dengan $x_i \in Grup_g$ dan $x_m \notin Grup_g$' (3.6)

Selanjutnya algoritma mencari titik-titik inti yang merupakan jangkauan kepadatan dari titik inti x_i, yaitu titik-titik inti yang merupakan jangkauan kepadatan langsung dari titik inti x_m. Sehingga algoritma ini melakukan pencarian titik-titik inti di dalam kumpulan tetangga terdekat tiap titik inti yang belum mempunyai grup secara rekursif. *Pseudocode* pembuatan grup titik inti tampak pada Algoritma 3.5, sedangkan *pseudocode* pencarian titik inti dalam tetangga terdekat sebuah titik ditunjukkan oleh Algoritma 3.6.

```
ALGORITMA Grupkan_Titik_Inti (Inti)
//Algoritma pembuatan grup titik inti
//Input   : list titik inti Inti
//Output: jumlah grup g
BEGIN
    g =1
    FOR setiap xi ∈ Inti , xi ∉ Grupg' DO
    BEGIN
        Buat grup baru, Grupg
        xi ∈ Grupg
        g = g + 1
        IF NN[xi] tidak kosong THEN Grupkan_Tetangga_Inti(xi, g)
    END
    Return g
END
```

Algoritma 3.5. *Pseudocode* Pengelompokkan Titik Inti

```
ALGORITMA Grupkan_Tetangga_Inti (y, g)
//Algoritma untuk mengelompokkan titik inti yang bertetangga dengan titik y
//Input   : titik y, indeks grup g
BEGIN
    FOR setiap x_m ∈NN[y] DO
        IF x_m ∈ Inti and x_m ∉ Grup_g' THEN
        BEGIN
            x_m ∈ Grup_g
            IF NN[x_im] tidak kosong THEN Grupkan_Tetangga_Inti(x_m, g)
        END
END
```

Algoritma 3.6. *Pseudocode* Pengelompokkan Titik Inti yang Saling Bertetangga

Algoritma ini melakukan perulangan sebanyak N_I kali dengan N_I adalah jumlah titik inti. Dalam setiap perulangannya algoritma ini memanggil algoritma Grupkan_Tetangga_Inti yang mempunyai kompleksitas waktu $O(N_I)$. Total kompleksitas waktu dari algoritma ini adalah $O(N_I{}^2)$.

3.4 Algoritma Pembentukan Titik Representasi

Setiap $grup_g$ yang terbentuk akan diwakili oleh sebuah titik representasi TR_g yang merupakan titik pusat dari grup tersebut. Pembentukan titik representasi ini dilakukan dengan mengambil nilai rerata setiap atribut dari masing-masing titik inti x_i yang tergabung dalam sebuah grup.

$$TR_g = \frac{\sum_{xi \in Grup_g} x_i}{|x_i|} \tag{3.7}$$

```
ALGORITMA Buat_TR (g)
//Algoritma pembuatan titik representasi
//Input : index grup g
BEGIN
    FOR setiap dimensi d DO
    BEGIN
        FOR setiap x_i ∈ Grup_g DO TR_g = TR_g + x_i
        TRg = TR_g / |x_i|
    END
END
```

Algoritma 3.7. *Pseudocode* Pembuatan Titik Representasi

Algoritma 3.7 menampilkan *pseudocode* dari pembuatan titik representasi. Dari *pseudocode* tersebut dapat diperkirakan kompleksitas waktunya adalah O(a*d) dengan a adalah notasi untuk menunjukkan jumlah anggota $grup_g$.

3.5 Algoritma Pembelahan Grup Titik Inti

Banyaknya grup titik inti yang terbentuk dalam metoda ini bergantung pada nilai *MinPts*. Semakin besar nilai *MinPts,* semakin luas radius kepadatan dari setiap titik inti yang terbentuk, dan semakin banyaklah jumlah titik inti yang berjangkauan kepadatan langsung dengan titik inti lain. Akibatnya grup titik inti yang dihasilkan menjadi sedikit. Bahkan tidak menutup kemungkinan jumlah grup inti yang terbentuk kurang dari jumlah klaster.

Jika hal itu terjadi, maka perlu dilakukan penambahan grup agar minimal sesuai dengan jumlah klaster. Penambahan grup ini dengan cara membelah grup yang beranggotakan titik inti terbanyak. Oleh karena itu, langkah awal adalah mengurutkan grup berdasarkan jumlah titik inti di dalamnya. Kemudian dibentuklah grup baru. Grup yang berbobot terberat $Grup_{top}$ kemudian dibelah titik-titik inti terhadap titik representasi grup tersebut. Titik-titik inti yang lebih besar dari titik representasi grup akan dipisahkan dan menjadi grup titik inti baru.

$$x_i \in \begin{cases} Grup_{top}, & \text{if } \|x_i\| < \|TR_{top}\| \\ grup\ baru\ Grup_g, & \text{if } \|x_i\| \geq \|TR_{top}\| \end{cases} \tag{3.8}$$

Jika ternyata grup dengan titik inti terbanyak tidak dapat dibelah, maka algoritma akan mencoba membelah grup dengan titik inti terbanyak kedua, dst. *Pseudocode* dari algoritma pembelahan grup titik inti ditunjukkan oleh Algoritma 3.8.

Kompleksitas waktu algoritma ini adalah penjumlahan dari kompleksitas waktu beberapa proses berikut:

1. Proses pengurutan grup membutuhan waktu $O(N_g^2)$
2. Proses pemindahan titik-titik membutuhkan waktu O(a*d) dengan notasi a adalah jumlah titik yang berada di $Grup_{top}$ ($|x_i|$, $x_i \in Grup_{top}$)
3. Proses pembuatan titik representasi untuk kedua grup yang memanggil algoritma Buat_TR, sesuai sub-bab 3.5 akan membutuhkan waktu O(a*d),

4. Perulangan jika grup tidak terbelah adalah maksimum sebanyak N_G kali untuk kasus terburuk (*worst-case*) dan 1 kali untuk kasus terbaik (*best-case*)

Total kompleksitas waktu dari algoritma modul ini adalah $O(N_G^2+N_G*a*d)$.

```
ALGORITMA Belah_Grup(NG)
//Algoritma pembelahan grup
//Input : jumlah grup nG
BEGIN
    //Urutkan grup-grup berdasarkan kepadatannya
    Urutkan Grup
    g = NG + 1
    Buat grup baru, Grupg
    top = 0
    DO
        FOR setiap xi ∈ Grup_top DO
            FOR setiap dimensi d dari xi DO
            BEGIN
                IF ||xi|| < ||TR_top|| THEN
                    xi ∈ Grupg
                ELSE
                    xi ∈ Grup_top
            END
        Buat_TR(top)
        Buat_TR(g)
        Top = top + 1
        //Jika grup tidak dapat dibelah, lakukan pembelahan pada grup selanjutnya
    WHILE Grup_top-1 tidak terbelah
END
```

Algoritma 3.8 *Pseudocode* Pembelahan Grup

3.6 Algoritma Penentuan Titik Pusat Awal Klaster

Penentuan titik pusat awal klaster dilakukan dengan menerapkan algoritma *k*-Means pada N_G titik representasi yang telah terbentuk, hanya jika jumlah grup yang terbentuk lebih banyak dari jumlah klaster ($N_G > K$). Jika jumlah grup sama dengan jumlah klaster ($N_G = K$), maka klasterisasi titik representasi tidak perlu dilakukan. Sebab dengan jumlah grup yang sama dengan jumlah klaster, semua titik representasi akan menjadi titik pusat klaster.

```
ALGORITMA Inisiasi_Titik_Pusat_Dari_TR(N_G,K)
//Algoritma penentuan titik pusat awal klaster dari titik representasi
//Input   : jumlah grup N_G, jumlah klaster k
BEGIN
    IF N_G > K THEN
    BEGIN
        //inisiasi k titik pusat klaster dari TR grup yang berbobot terberat
        Urutkan Grup
        j = 1
        WHILE j ≤ K DO
        BEGIN
            g = j
            FOR setiap dimensi d dari v_j DO v_j = TR_g, j,g ∈{1,...,K}, g ∈{1,..., N_G }
            j = j + 1
        END
        DO
        //Masukkan TR_g dalam klaster C_j terdekatnya
            FOR setiap TR_g, g ∈{1,..., N_G } DO
                FOR setiap v_j, j ∈{1,...,K} DO
                BEGIN
                    FOR setiap dimensi d dari v_j DO
                        d(TR_g, v_j) = √(Σ(TR_g − v_j)^2)
                    //Jika adalah jarak minimum, TRg anggota klaster Cj
                    IF d(TR_g,v_j) < d(TR_g,v_j') THEN TR_g ∈ C_j
                END
            //Geser posisi titik pusat klaster
            FOR setiap v_j, j ∈{1,...,k} DO
                FOR setiap dimensi d dari v_j DO
                BEGIN
                    FOR setiap TR_g ∈C_j, g ∈{1,..., N_G } DO v_j = v_j + TR_g
                    v_j = v_j / |TR_g|
                END
        WHILE terdapat v_j yang berubah posisi
        //Tempatkan setiap titik inti di Grupg ke klaster Cj
        FOR setiap x_i ∈ Grup_g DO x_i ∈ C_j, dengan TR_g ∈Cj, j ∈{1,...,k}, g ∈{1,..., N_G }
    END
    ELSE //jumlah grup sama dengan jumlah klaster
        j = 1
        WHILE j ≤ K
        BEGIN
            g = j
            v_j = TR_g, j ∈{1,...,k}, g ∈{1,...,nG}
        END
        FOR setiap x_i ∈ Grup_g DO x_i ∈ C_j, dengan g = j
END
```

Algoritma 3.9. *Pseudocode* Algoritma Penentuan Titik Pusat Awal Klaster.

Jika jumlah grup titik inti lebih banyak dari jumlah klaster, maka langkah-langkah algoritma ini adalah :

1. Inisiasi k titik pusat klaster v_j yang diambil dari titik representasi TR_g.

Tidak seperti pada algoritma *k*-Means asli maupun algoritma k-Means berpartisi sederhana, inisiasi k titik pusat klaster di sini tidak mengambil secara sembarang k titik representasi, melainkan mengambil berdasarkan titik representasi terberat (yang paling banyak mewakili titik inti). Langkah tersebut dilakukan dengan pertimbangan bahwa semakin banyak titik inti yang berkumpul, maka semakin besar kemungkinan kumpulan tersebut adalah klaster. Karena itu, sebelumnya dilakukan pengurutan pada grup-grup titik inti berdasarkan bobotnya. Setelah itu k titik representasi teratas dari hasil pengurutan tersebut dipakai sebagai inisiasi awal titik pusat klaster.

2. Penentuan klaster bagi tiap titik representasi dengan cara mencari jarak Euclidean *d* terdekat antara titik representasi TR_g dengan titik pusat klaster v_j.

   $TR_g \in C_j$ dimana $d(TR_g, v_j) < d(TR_g, v_j')$ (3.9)

3. Setelah semua titik representasi TR_g berada dalam sebuah klaster C_j, maka dilakukan perubahan posisi titik pusat klaster v_j sehingga berada di pusat klaster.

$$v_j = \frac{\sum_{TRg \in Cj} TR_g}{|TR_g|} \tag{3.10}$$

4. Selama terjadi perubahan posisi titik pusat klaster v_j, maka algoritma mengulangi langkah 2 dan 3.
5. Penempatan titik inti ke dalam klaster.

   Setelah semua titik representasi berada dalam klaster yang tepat, algoritma kemudian akan menempatkan titik-titik inti x_i kedalam klaster dimana titik representasi dari grupnya TR_g berada.

   $x_i \in C_j$ dimana $x_i \in Grup_g$ dan $TR_g \in C_j$ (3.11)

Apabila jumlah grup titik inti sama dengan jumlah klaster ($g = K$), maka algoritma hanya perlu mengeset titik representasi sebagai titik pusat klaster dan menempatkan titik-titik inti x_i di $grup_g$ ke dalam klaster C_j.

$v_j = TR_g$ dimana $g=j$ (3.12)

$x_i \in C_j$ dimana $x_i \in Grup_g$ dan $g=j$ (3.13)

Kompleksitas waktu dari algoritma ini dibagi-bagi menjadi :

1. Pengurutan grup dengan kompleksitas $O(N_g^2)$.

2. Inisiasi awal titik pusat klaster membutuhkan waktu O(k*d)
3. Penempatan titik representasi dalam klaster dan penggeseran posisi titik pusat klaster mempunyai total kompleksitas $O(i_{TR}*k*\ N_g\ *d)$ dengan i menunjukkan jumlah iterasi. Karena *k*-Means dilakukan pada titik representasi yang jumlah jauh lebih sedikit dari jumlah titik data, maka jumlah iterasi yang terjadi pun akan sangat sedikit. Hal ini akan dibuktikan pada uji coba pada sub-bab 5.4.1
4. Penempatan titik inti ke dalam klaster membutuhkan waktu $O(N_I)$ dengan notasi N_I adalah jumlah titik inti.
5. Untuk kondisi jumlah grup sama dengan jumlah klaster dibutuhkan waktu untuk mengeset titik pusat dan penempatan titik inti ke dalam klaster sebanyak $O(k*d + N_I)$.

Sehingga total kompleksitas waktunya adalah $O(N_g^2+(k*d)*(1+i_{TR}*N_g)+N_I)$ dengan komputasi waktu untuk kasus terbaik (*best-case*) $O(k*d + N_I)$.

3.7 Algoritma Penentuan Akhir Titik Pusat Klaster dan Klasterisasi Titik Data

Langkah terakhir dari metoda penelitian ini adalah menentukan posisi akhir titik pusat klaster dan melakukan klasterisasi seluruh titik data dengan menggunakan titik pusat yang telah dihasilkan Algoritma 3.9 sebagai inisiasi. Algoritma ini hanya akan dilakukan jika terdapat titik tepi pada sebaran titik data. Sebab, jika tidak ada titik tepi, maka seluruh titik data (yang berarti seluruhnya adalah titik inti) telah diklasterkan oleh algoritma sebelumnya.

Langkah-langkah dari klasterisasi akhir yang *pseudocode*-nya seperti pada Algoritma 3.10 ini sebagai berikut:

1. Penempatan titik-titik data x_i ke dalam sebuah klaster C_j yang berjarak Euclidean *d* terdekat dengannya.

   $x_i \in C_j$ dimana $d(x_i, v_j) < d(x_i, v_j'); x_i \in X$ (3.14)

2. Perubahan posisi titik pusat klaster v_j sehingga berada di pusat titik-titik yang menjadi anggota klaster C_j.

$$v_j = \frac{\sum x_i}{|x_i|} \quad (3.15)$$

3. Ulangi langkah 1 dan 2 selama terjadi perubahan posisi titik pusat klaster v_j.

Kompleksitas waktu dari ini adalah $O(i_a*N*k*d)$ untuk penempatan titik ke dalam klaster dan pergeseran posisi titik pusat klaster dengan notasi i_a adalah jumlah iterasi klasterisasi akhir, n adalah jumlah titik data, k adalah jumlah klaster dan d adalah dimensi / atribut. Walau terlihat memiliki kompleksitas waktu yang tinggi, namun sebenarnya klasterisasi akhir ini hanya memerlukan beberapa kali iterasi. Hal itu disebabkan klasterisasi akhir ini menggunakan titik-titik pusat awal klaster yang dihasilkan dari klasterisasi grup-grup titik inti (seperti pada sub-bab 3.6), sehingga besar kemungkinan titik-titik pusat itu telah berada di pusat klaster.

```
ALGORITMA kMeans_Akhir
//Algoritma penentuan akhir titik pusat klaster dan klasterisasi data
BEGIN
    IF N_T >0 THEN
    BEGIN
        DO
        //Masukkan titik x_i dalam klaster C_j terdekatnya
            FOR setiap x_i, i ∈{1,...,N} DO
                FOR setiap v_j, j ∈{1,...,K} DO
                    FOR setiap dimensi d dari v_j DO
                    BEGIN
                        d(x_i, v_j) = √(Σ(x_i - v_j)^2)
                        //Jika adalah jarak minimum, x_i anggota klaster Cj
                        IF d(x_i,v_j) < d(x_i,v_j') THEN x_i ∈ C_j
                    END
            //Geser posisi titik pusat klaster
            FOR setiap v_j, j ∈{1,...,K} DO
                FOR setiap dimensi d DO
                BEGIN
                    FOR setiap x_i ∈ C_j DO v_j = v_j + x_i
                    v_j = v_j / | x_i|
                END
        WHILE terdapat v_j yang berubah posisi
    END
END
```

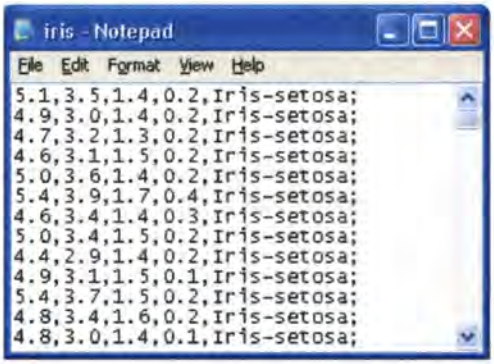
Algoritma 3.10. *Pseudocode* Agoritma Penentuan Akhir Titik Pusat Klaster dan Klasterisasi Seluruh Titik Data

BAB 4
IMPLEMENTASI ALGORITMA

Algoritma yang telah dibahas pada bab 3 diimplementasikan dalam bahasa pemrograman Basic dengan platform .Net. Sebagai awalan, sub-bab 4.1. menjelaskan cara penyimpanan set data, secara eksternal dan dalam program. Sedangkan sub-bab 4.2 sampai sub-bab 4.7. merupakan penjelasan implementasi desain algoritma seperti yang telah diterangkan di bab 3.

4.1 Penyimpanan Set Data

Pada implementasi ini, set data yang akan diklaster diambil dari file teks dengan ekstensi .data. Satu data dengan data yang lain dipisahkan dengan tanda ';' (titik koma), sedangkan atribut-atribut masing-masing data dipisahkan dengan tanda ',' (koma). Gambar 4.1 adalah contoh format penyimpanan data.



Gambar 4.1. Format Penyimpanan Data

Pengambilan set data tersebut dilakukan dengan membaca rangkaian character dalam sebuah file teks. Untuk itu program membutuhkan *library* tambahan yaitu *namespace* System.IO. Dengan *namespace* tersebut, program dapat membuat sebuah objek dari kelas *StreamReader* yang berfungsi untuk membaca isi file teks dan menempatkannya pada sebuah variabel string, strData, dengan fungsi *ReadToEnd*.

Nilai string yang diambil itu adalah keseluruhan set data. Untuk memisahkan satu data dengan yang lain digunakan fungsi *Split*. Fungsi *Split*

adalah fungsi milik kelas String pada .Net yang mengambil bagian-bagian dari sebuah string (*sub-string*) yang dipisahkan oleh karakter tertentu. Masing-masing bagian dari string tersebut diletakkan dalam sebuah array. Penelitian ini banyak memanfaatkan fungsi ini dalam implementasi algoritmanya. Untuk pemisahan antar data, parameter pemisah yang digunakan adalah tanda ";" (titik koma). Sedangkan untuk pemisahan atribut-atribut dari setiap data, fungsi *Split* kembali digunakan dengan tanda "," (koma) sebagai parameter pemisah.

Program kemudian membuat sebuah array 1 dimensi, bernama Titik, dengan panjang sejumlah data. Array tersebut bertipe **Point**. Kelas Point dibuat untuk menyimpan segala informasi berkaitan dengan titik-titik data. Properti dari kelas Point ditunjukkan oleh Tabel 4.1. Sedangkan kode program untuk membaca data ditunjukkan pada Segmen Program 4.1 berikut ini :

```
Sub ReadData()
    Dim strDBName As String
    Dim strData As String
    Dim i, j As Integer
    Dim objReader As StreamReader
ulang:
    Console.Write("Nama File : ")
    strDBName = Console.ReadLine
    Console.WriteLine()
    Try
    objReader = New StreamReader("dataset\" & strDBName)
       strData = objReader.ReadToEnd
       objReader.Close()
       objReader = Nothing
    Catch ex As Exception
    Console.WriteLine(ex)
       objReader = Nothing
       GoTo ulang
    End Try

    Dim strRec() As String = strData.Split(";")
    ReDim Titik(strRec.GetUpperBound(0) - 1)

    For i = 0 To strRec.GetUpperBound(0) - 1
    Titik(i) = New Point
       Titik(i).Set_Attributes(strRec(i))
    Next
End Sub
```

Segmen Program 4.2. Fungsi *ReadData*.

Tabel 4.1. Properti dari kelas Point

No.	Nama	Tipe Data	Keterangan
1	RecNum	String	properti untuk menyimpan posisi titik dalam set data adalah titik ke berapa
2	Attributes()	Double	properti bertipe array 1 dimensi yang menyimpan nilai masing-masing atribut sebuah titik
3	Label	String	properti yang menyimpan kelas/klaster dari titik
4	Cluster	String	properti untuk menyimpan keluaran dari algoritma, yaitu pada klaster mana sebuah titik ditempatkan
5	NN	String	singkatan dari *Nearest Neighbor* adalah properti yang menyimpan RecNum tetangga terdekat dari tiap titik. Antar tetangga satu dengan yang lain dipisahkan dengan tanda ; (titik koma)
6	ClosestPoint	String	properti yang menyimpan RecNum titik tetangga terdekat kesatu dari tiap titik
7	FurthestPoint	String	properti yang menyimpan RecNum titik tetangga terdekat ke-n dari tiap titik
8	Dmin	Double	properti yang menyimpan jarak antara titik dengan ClosestPoint-nya
9	Eps	Double	properti yang menyimpan jarak antara titik dengan FurthestPoint-nya
10	isInti	Boolean	properti untuk menandai apakah titik tersebut titik inti atau titik tepi. Jika titik inti maka properti ini bernilai TRUE, jika titik tepi bernilai FALSE
11	DependentOf	String	properti yang menyimpan RecNum titik-titik yang merujuk pada sebuah titik. Antar satu titik dengan titik lain dipisahkan dengan tanda ; (titik koma)
12	Group	String	properti yang menyimpan termasuk dalam grup titik inti manakah sebuah titik (jika titik adalah titik inti). Jika titik adalah titik tepi, maka grup bernilai '-' (garis mendatar)

Selain dua belas properti di atas, kelas Point juga memiliki sebuah method *Set_Attributes* yang berfungsi untuk menyimpan nilai-nilai atribut tiap titik ke properti attributes(). Kode method *Set_Attributes* ditunjukkan pada Segmen Program 4.2.

```
Public Sub Set_Attributes(ByVal str As String)
    Dim strkol() As String = str.Split(",")
    ReDim _attr(strkol.GetUpperBound(0) - 1)
    Dim d As Integer
    _dimensi = strkol.GetUpperBound(0)
    For d = 0 To strkol.GetUpperBound(0) - 1
    _attr(d) = strkol(d)
    Next
        _label = strkol(strkol.GetUpperBound(0))
End Sub
```

Segmen Program 4.2. Method *Set_Attributes* pada Kelas Point

4.2 Pencarian Titik Inti

Sesuai dengan algoritma pada Sub Bab 3.2, terdapat dua langkah dalam implementasi pencarian titik inti yaitu pencarian tetangga terdekat dan penentuan titik inti.

4.2.1 Pencarian Tetangga Terdekat

Pencarian tetangga terdekat setiap titik data dilakukan dengan memanggil fungsi *FindNeighbors* seperti pada Segmen Program 4.3. Setelah mencari tetangga terdekat seluruh titik, maka dihitung rerata radius kepadatan.

```
While rec <= Titik.GetUpperBound(0)
    Titik(rec).RecNum = CStr(rec)
    FindNeighbors(rec)
    sumEps += Titik(rec).Eps
    rec += 1
End While
AvgEps = Format(sumEps / rec, "0.00")
```

Segmen Program 4.3. Pencarian Tetangga Terdekat

Fungsi *FindNeighbor*, ditunjukkan oleh Segmen Program 4.5, mempunyai argumen rec1 yang merupakan RecNum titik yang sedang dicari tetangga terdekatnya. Fungsi ini menyediakan beberapa array 1 dimensi, yaitu 1) *d* untuk menyimpan jarak titik dengan semua titik data yang lain; 2) *Neighbors* untuk menyimpan *MinPts*-1 RecNum tetangga terdekat dari titik; 3) *Jarak* untuk menyimpan jarak titik dengan masing-masing tetangga terdekatnya. Variabel idx berfungsi sebagai *counter* sekaligus penentu index array Neighbor.

Pada awalnya, jarak maksimum (*max*) diset 0, sedangkan jarak minimum (*min*) diset nilai terbesar. Lalu, untuk setiap $titik_{rec}$, jika bukan titik yang bersangkutan (rec <> rec1) dilakukan langkah-langkah berikut:

1. Pencarian jarak Euclidean antar kedua titik dengan memanggil fungsi Euclidean :

```
Function Euclidean(ByVal x As Integer,ByVal y As Integer) As Single
    Dim i As Integer
    Dim tem, tem1 As Single
    For i = 0 To Dimensi - 1
    tem = Titik(x).Attributes(i) - Titik(y).Attributes(i)
       tem1 += (tem * tem)
    Next
    Return Math.Sqrt(tem1)
End Function
```

Segmen Program 4.4. Fungsi *Euclidean*

2. Jika masih terdapat tempat di array Neighbor (idx < *MinPts*-1), maka
   a. $Titik_{rec}$ adalah tetangga terdekat ke-idx dan simpan jaraknya.
   b. Jika jarak tersebut kurang dari jarak minimum, maka set Neighbor(idx) sebagai tetangga terdekat pertama (minId = idx) dan simpan pula jaraknya sebagai jarak minimum.
   c. Jika jarak tersebut lebih besar dari jarak maksimum, maka set Neighbor(idx) sebagai tetangga terdekat ke-n (maxId = idx) dan simpan pula jaraknya sebagai jarak maksimum.
   d. Tambahkan nilai idx dengan 1.
3. Jika array Neighbor sudah penuh, maka bila jarak $titik_{rec}$ lebih besar dari jarak maksimum, maka $titik_{rec}$ bukan tetangga terdekat. Jika jarak $titik_{rec}$ masih kurang dari jarak maksimum, maka :
   a. Gantikan tetangga terdekat ke-n dengan $titik_{rec}$ (Neighbor(maxId) = rec). simpan pula jaraknya.
   b. Urutkan tetangga terdekat dari jarak terkecil hingga jarak terjauh.
   c. Perbarui tetangga terdekat pertama dan ke-n sesuai hasil urutan. Simpan pula jarak masing-masing.

```
Sub FindNeighbors(ByVal rec1 As Long)
   Dim rec, idx As Long
   Dim maxId, minId, j As Byte
   Dim d(Titik.GetUpperBound(0)), jarak(MinPts - 2) As Single
   Dim Neighbor(MinPts - 2) As String
   Dim max As Single = 0.0
    Dim max As Single = min.MaxValue
   For rec = 0 To Titik.GetUpperBound(0)
   If rec <> rec1 Then
      d(rec) = Format(Euclidean(rec1, rec), "0.00")
        If idx < MinPts - 1 Then
         Neighbor(idx) = rec
           jarak(idx) = d(rec)
           If d(rec) < min Then
            min = jarak(idx)
             minId = idx
           End If
           If d(rec) > max Then
            max = jarak(idx)
            maxId = idx
           End If
           idx += 1
            Else 'jika jarak lebih, maka bukan neigbour
         If d(rec) < max Then
            Dim i, k As Byte
              Dim tem As Long
              Neighbor(maxId) = rec
              jarak(maxId) = d(rec)
                ''sort
              For i = 0 To MinPts - 3
                For k = i + 1 To MinPts - 2
                    If jarak(i) > jarak(k) Then
                         tem = jarak(k)
                         jarak(k) = jarak(i)
                         jarak(i) = tem
                         tem = CDbl(Neighbor(k))
                         Neighbor(k) = Neighbor(i)
                         Neighbor(i) = CStr(tem)
                      End If
                  Next
              Next
              min = jarak(0)
              max = jarak(MinPts - 2)
              minId = 0
              maxId = MinPts - 2
            End If
         End If
      End If
   Next
   For j = 0 To MinPts - 2
   Titik(rec1).NN = Titik(rec1).NN & Neighbor(j) & ";"
   Next
   Titik(rec1).dMin = min
   Titik(rec1).ClosestPoint = Neighbor(minId)
   Titik(rec1).Eps = max
```

Segmen Program 4.5. Fungsi *FindNeighbors*

4. Jika pencarian tetangga terdekat telah selesai, maka simpan RecNum dari tetangga-tetangga terdekat itu dalam properti NN. Simpan pula RecNum

tetangga terdekat pertama sebagai ClosestPoint, jaraknya sebagai dmin, RecNum tetangga terdekat ke-n sebagai FurthestPoint dan jaraknya sebagai Eps.

4.2.2 Penentuan Titik Inti

Langkah ini dilakukan dengan sekali lagi menelusuri setiap titik data. Untuk masing-masing $titik_i$, dibuatlah sebuah objek *close* yang merupakan *ClosestPoint* $titik_j$.

Jika dmin dari $titik_i$ kurang dari / sama dengan Eps dari close dan Eps dari close kurang dari rerata Eps maka $titik_i$ adalah titik inti dan tambahkan $titik_i$ sebagai salah satu titik yang merujuk pada close.

Jika salah satu kondisi di atas tidak terpenuhi, maka $titik_i$ bukan titik inti. Karena $titik_i$ bukan titik inti, maka seluruh titik yang merujuk padanya (jika ada) juga bukanlah titik inti. Perubahan pada titik-titik rujukan dari $titik_i$ tersebut dilakukan dengan memanggil fungsi *ChangeDependeeToTepi* yang kode programnya seperti pada Segmen Program 4.7. Keseluruhan kode program unutk penentuan titik inti adalah seperti pada Segmen Program 4.6.

```
Dim close As Point
Dim i As Integer
For i = 0 To Titik.GetUpperBound(0)
    close = Titik(Titik(i).ClosestPoint)
    If Titik(i).dMin <= close.Eps And close.Eps <= AvgEps Then
    Titik(i).isInti = True
       close.DependentOf += i & ";"
       strCore += i & ";"
    Else
    Titik(i).isInti = False
       Titik(i).Group = "-"
        Ntepi += 1
       If Titik(i).DependentOf <> "" Then
        Dim strAnak() As String = Titik(i).DependentOf.Split(";")
          ChangeDependeeToTepi(strAnak)
       End If
    End If
   close = Nothing
Next
```

Segmen Program 4.6. Penentuan Titik Inti.

```
Sub ChangeDependeeToTepi(ByVal str() As String)
    Dim i As Integer
    Dim strsplit as string
    For i = 0 To str.GetUpperBound(0) - 1
    If Titik(str(i)).isInti = True Then
        Titik(str(i)).isInti = False
          Titik(str(i)).Group = "-"
           Ntepi += 1
          If Titik(str(i)).DependentOf <> "" Then
              strsplit = Titik(str(i)).DependentOf
              ChangeDependeeToTepi(strsplit.Split(";"))
           End If
        End If
    Next
End Sub
```

Segmen Progarm 4.7. Fungsi *ChangeDependeeToTepi*

Argumen str() pada fungsi *ChangeDependeeToTepi* berisikan RecNum dari titik-titik yang merujuk pada $titik_i$ pada program pemanggilnya. Jika titik-titik tersebut sebelumnya adalah titik inti, maka fungsi ini akan mengubahnya menjadi titik tepi (bukan titik inti). Lalu, apabila titik tersebut juga merupakan rujukan dari titik lain, maka fungsi ini kembali dipanggil.

4.3 Pembentukan Grup Titik Inti

Selain kelas Point, dalam program ini juga terdapat kelas Group yang berfungsi menyimpan informasi tentang grup-grup titik inti yang terbentuk. Properti dari kelas Group ditunjukkan oleh Tabel 4.2.

Tabel 4.2. Properti kelas Group

No.	Properti	Tipe Data	Keterangan
1	Anggota	String	properti untuk menyimpan titik-titik inti mana saja yang masuk dalam grup tersebut. Antar titik inti dipisahkan dengan tanda ; (titik koma)
2	Jumlah	Long	properti untuk menyimpan jumlah titik inti dalam grup tersebut
3	TRep( )	Double	properti untuk menyimpan posisi titik representasi dari grup yang bersangkutan
4	Cluster	String	properti yang menyimpan pada klaster mana titik representasi dari grup ini berada

Pembentukan grup dilakukan dengan menelusuri setiap titik inti. Jika $Inti_i$ belum mempunyai grup, maka dibentuklah grup baru $grup_g$ dan $Inti_i$ menjadi

anggota $grup_g$ tersebut. Langkah selanjutnya adalah mencari apakah terdapat titik inti lain yang menjadi tetangga terdekat dari $Inti_i$. Pencarian ini memanggil fungsi DR yang merupakan singkatan dari *Directly Reachable*. Jika terdapat titik inti yang bertetangga dengan $Inti_i$ dan belum mempunyai grup, maka masukkan titik inti tetangga itu, sebutlah sebagai $Inti_j$ dalam $grup_g$. Secara rekursif, fungsi DR akan memasukkan titik inti tetangga dari $Inti_j$ (jika ada) ke dalam grup yang sama, dan seterusnya. Kode program pembentukan grup ditunjukkan pada Segmen Program 4.8., sedangkan fungsi DR pada Segmen Program 4.9.

```
Dim g As Integer
Dim Inti() as string
Inti = strCore.Split(";")
For i = 0 To Inti.GetUpperBound(0) - 1
   If Titik(Inti (i)).Group = "" Then
   If g > grup.GetUpperBound(0) Then ReDim Preserve grup(g)
      grup(g) = New Group
      Titik(Inti(i)).Group = g
      grup(g).Anggota += Inti(i) & ";"
      grup(g).Jumlah += 1
      If Titik(Inti(i)).NN.Length > 0 Then DR(Inti(i)), g)
      g += 1
   End If
Next
```

Segmen Program 4.8. Pembentukan Grup Titik Inti

```
Sub DR(ByVal y As String, ByVal No As Integer)
   Dim j As Integer
   Dim strnn() As String = Titik(y).NN.Split(";")
   For j = 0 To strnn.GetUpperBound(0) - 1
      If Titik(CInt(strnn(j))).isInti = True Then
         If (Titik(CInt(strnn(j))).Group = "") Then
            Titik(CInt(strnn(j))).Group = No
            grup(No).Anggota += strnn(j) & ";"
            grup(No).Jumlah += 1
            If Titik(CInt(strnn(j))).NN.Length > 0 Then _
               DR(strnn(j), No)
         End If
      End If
   Next
End Sub
```

Segmen Program 4.9. Fungsi DR.

4.4 Pembuatan Titik Representasi Grup

Pembuatan titik representasi sebuah grup $TRep_r$ dilakukan dengan memanggil method *Make_Rep_Point* dari $grup_r$. Method ini menentukan posisi

dari $Trep_r$ dengan menghitung rerata nilai aatribut titik-titik inti yang menjadi anggota $grup_r$. Di bawah ini adalah segmen program untuk pembuatan seluruh titik representasi (Segmen Program 4.10.) dan Method Make_Rep_Point (Segmen Program 4.11)

```
For r = 0 To g - 1
    arr = Nothing
    arr = grup(i).Anggota.Split(";")
    grup(r). Make_Rep_Point(Titik, Dimensi, arr)
Next
```

Segmen Program 4.10. Pembuatan Seluruh Titik Representasi

```
Public Sub Make_Rep_Point(ByVal dt As Object, _
ByVal jumCol As Integer, ByVal str As String())
    Dim j, k As Integer
    ReDim _trep(jumCol - 1)
    For j = 0 To jumCol - 1
     For k = 0 To str.GetUpperBound(0) - 1
         _trep(j) += dt(str(k)).Attributes(j)
        Next
       _trep(j) = Format(_trep(j) / (str.Length - 1), "0.00")
    Next
End Sub
```

Segmen Program 4.11. Method *Make_Rep_Point* pada Kelas Group

4.5 Pembelahan Grup Titik Inti

Seperti yang telah dibahas di Sub Bab 3.5, grup yang dibelah adalah grup yang memiliki bobot terberat. Oleh karena itu grup-grup yang telah dibentuk perlu diurutkan berdasarkan jumlah titik inti di dalamnya, yaitu dari yang terberat hingga teringan. Dalam implementasi ini digunakan metode pengurutan *InsertionSort* yang diimplemetasikan dalam fungsi *Sort_Group_Den* seperti pada Segmen Program 4.12.

```
Sub Sort_Group_Den(ByVal jml As Integer, Optional ByVal tipe As String
= "ASC")
    Dim tem As Group
    Dim i, j As Integer
    If tipe.ToUpper = "DESC" Then
    j = 0
       For i = 1 To jml - 1
        tem = New Group
          tem = grup(i)
          j = i - 1
          While j >= 0
           If grup(j).Jumlah >= tem.Jumlah Then Exit While
             grup(j + 1) = grup(j)
             j = j - 1
           End While
          grup(j + 1) = tem
          tem = Nothing
        Next
    Else
    j = 0
       For i = 1 To jml - 1
        tem = New Group
          tem = grup(i)
          j = i - 1
          While j >= 0
           If grup(j).Jumlah <= tem.Jumlah Then Exit While
             grup(j + 1) = grup(j)
             j = j - 1
           End While
          grup(j + 1) = tem
          tem = Nothing
        Next
    End If
End Sub
```

Segmen Program 4.12. Fungsi *Sort_Group_Den*

Setelah diurutkan, maka grup baru, $grup_g$, dibentuk. Lalu pembelahan $grup_{MaxG}$ sebagai grup terberat pun dimulai. Dengan nilai MaxG dimulai dari 0 (nol), untuk setiap titik inti dalam $grup_{MaxG}$ dilihat apakah lebih besar daripada $Trep_{MaxG}$ atau tidak. Titik-titik inti yang lebih besar dari $TRep_{MaxG}$ kemudian ditempatkan pada $grup_g$ sedangkan yang lebih kecil tetap dalam $grup_{MaxG}$. Segmen Program 4.13 menampilkan bagian kode program pembelahan grup ini.

```
While g < K 'jika jumlah grup < jumlah cluster yg diinginkan
    maxG = 0
ulang_belah:
    Sort_Group_Den(g, "desc")
    ReDim Preserve grup(grup.GetUpperBound(0) + 1)
    Dim d As Integer
    grup(grup.GetUpperBound(0)) = New Group
    arr = Nothing
    arr = grup(maxG).Anggota.Split(";")
    grup(maxG).Anggota = ""
    grup(maxG).Jumlah = 0
    Dim sem1, sem2, sem, tem, tem1, tem2 As Single
    For i = 0 To arr.GetUpperBound(0) - 1
    sem2 = 0
       tem2 = 0
       For d = 0 To centroid.GetUpperBound(1)
        sem1 = Titik(arr(i)).Attributes(d)
          tem1 = grup(maxG).TRep(d)
          tem2 += tem1 * tem1
          sem2 += sem1 * sem1
        Next
       sem = Math.Sqrt(sem2)
       tem = Math.Sqrt(tem2)
        If sem > tem Then
        Titik(arr(i)).Group = grup.GetUpperBound(0)
          grup(grup.GetUpperBound(0)).Anggota += arr(i) & ";"
          grup(grup.GetUpperBound(0)).Jumlah += 1
        Else
        grup(maxG).Anggota += arr(i) & ";"
          grup(maxG).Jumlah += 1
        End If
    Next
```

Segmen Program 4.13 Pembelahan Grup berdasarkan letak titik terhadap Trep

Apabila ternyata $grup_{MaxG}$ tidak terbelah, semua titik inti anggotanya tidak ada yang pisah, maka MaxG ditambah 1, lalu mengulangi proses pembelahan di atas.. Kode program yang menangani kondisi ini ditunjukkan oleh Segmen Program 4.14.

```
If grup(grup.GetUpperBound(0)).Jumlah = 0 Or grup(maxG).Jumlah = 0 Then
    If grup(maxG).Jumlah = 0 Then
        arr = Nothing
        arr = grup(grup.GetUpperBound(0)).Anggota.Split(";")
        grup(grup.GetUpperBound(0)).Make_Rep_Point(Titik,Dimensi,arr)
        For i = maxG To grup.GetUpperBound(0) - 1
            grup(i) = grup(i + 1)
        Next
    ElseIf grup(grup.GetUpperBound(0)).Jumlah = 0 Then
        arr = Nothing
        arr = grup(maxG).Anggota.Split(";")
        grup(maxG).Make_Rep_Point(Titik, Dimensi, arr)
    End If
    ReDim Preserve grup(grup.GetUpperBound(0) - 1)
    maxG += 1
    GoTo ulang belah
```

Segmen Program 4.14 Jika $Grup_{MaxG}$ tidak terbelah

Jika grup terbelah, maka langkah selanjutnya adalah membuat titik representasi bagi kedua grup, seperti pada Segmen Program 4.15.

```
Else
    g = g + 1
    arr = Nothing
    arr = grup(maxG).Anggota.Split(";")
    grup(maxG).Make_Rep_Point(Titik, Dimensi, arr)
    arr = Nothing
    arr = grup(grup.GetUpperBound(0)).Anggota.Split(";")
    grup(grup.GetUpperBound(0))._
    Make_Rep_Point(Titik,Dimensi,arr)
End If
End While
```

Segmen Program 4.15 Pembuatan Titik Representasi untuk Kedua Grup

4.6 Penerapan K-Means pada Titik Representasi

Implementasi klasterisasi k-Means pada titik representasi diterapkan dalam langkah-langkah berikut:

4.6.1 Inisiasi Titik Pusat Klaster

Inisiasi titik pusat klaster dilakukan dengan mengambil k titik representasi terberat. Implementasi dari metoda ini adalah dengan mengurutkan terlebih dahulu grup-grup titik inti dari yang terberat hingga teringan dengan menggunakan kode program pada Segmen Program 4.12. Setelah diurutkan, maka diambil k titik representasi ($TRep_0$, $TRep_1$, ... $TRep_k$) sebagai titik pusat klaster, $centroid_j$, seperti pada Segmen Program 4.16. berikut:

```
If g > K Then
    'inisiasi centroid
    Sort_Group_Den(g, "desc")
    For i = 0 To K - 1
    For j = 0 To centroid.GetUpperBound(1)
       centroid(i, j) = grup(i).TRep(j)
      Next
    Next
```

Segmen Program 4.16. Inisiasi Titik Pusat Klaster.

4.6.2 Pencarian Klaster Terdekat bagi Setiap Titik Representasi

Implementasi pencarian klaster terdekat untuk setiap titik representasi adalah sebagai berikut :

```
For i = 0 To g - 1
    jarakMin = Format(Single.MaxValue, "0.00")
    For j = 0 To K - 1
    jarak = Format(EuTRvsCentro(centroid,i,j,Dimensi,"0.00")
       If jarakMin > jarak Then
        jarakMin = jarak
          grup(i).Cluster = j
        End If
    Next
    AnakCentro(grup(i).Cluster) += i & ";"
Next
```

Segmen Program 4.17. Pencarian Klaster Terdekat

Untuk setiap titik representasi grup $TRep_g$ dihitung jarak Euclideannya dengan sebuah titik pusat klaster $centroid_j$. jika jarak tersebut lebih kecil dari jarak minimal, maka $TRep_g$ adalah anggota klaster C_j dan jarak antara keduanya adalah jarak minimal dari $TRep_g$. Perhitungan jarak Euclidean dilakukan dengan memanggil fungsi *EuTRvsCentro* berikut :

```
Function EuTRvsCentro(ByVal dt2 As Object, ByVal x As Integer, ByVal y
As Integer, ByVal dimensi As Integer) As Single
    Dim i As Integer
    Dim tem, tem1 As Single

    For i = 0 To dimensi - 1
    tem = grup(x).TRep(i) - dt2(y, i)
      tem1 += (tem * tem)
    Next
    Return Math.Sqrt(tem1)
End Function
```

Segmen Program 4.18. Fungsi EuTRvsCentro

4.6.3 Pergeseran Posisi Titik Pusat Klaster

Agar titik pusat klaster $centroid_j$ berada di tengah, digunakan kode program seperti Segmen Program 4.19. Untuk masing-masing klaster, nilai atribut-atribut setiap titik di dalamnya dijumlahkan. Hasil penjumlahan itu kemudian dibagi dengan jumlah titik dalam sebuah klaster hingga menghasilkan posisi baru dalam dimensi tertentu. Posisi tersebut kemudian dibandingkan dengan posisi titik pusat yang sekarang. Jika berubah, maka ubah posisi titik pusat dengan posisi yang baru dan beri tanda bahwa terjadi perubahan.

Sub Bab 4.6.2 dan 4.6.3 diulang selama terjadi pergeseran posisi titik pusat klaster (ismove = TRUE).

```
Dim d, r As Integer
Dim newCentro(centroid.GetUpperBound(0), Dimensi - 1) As Single

For j = 0 To K - 1
    arr = Nothing
   arr = AnakCentro(j).Split(";")
   For d = 0 To centroid.GetUpperBound(1)
    For i = 0 To arr.GetUpperBound(0) - 1
    newCentro(j, d) += grup(arr(i)).TRep(d)
    Next
       newCentro(j,d) = newCentro(j,d)/(arr.Length-1)
       If newCentro(j, d) <> centroid(j, d) Then
       ismove = True
          centroid(j, d) = newCentro(j, d)
       End If
    Next
Next
```

Segmen Program 4.19. Pergeseran Posisi Titik Pusat Klaster

4.6.4 Penempatan Titik Inti dalam Klaster

Setelah klaster terbentuk, yaitu titik-titik pusat klaster tak lagi berubah posisi, maka dilakukan penempatan titik inti dalam klaster. Untuk setiap titik pusat klaster $centroid_j$, jika terdapat titik representasi $TRep_g$, maka seluruh titik inti yang berada dalam $grup_g$ dimasukkan ke dalam $klaster_j$. Segmen Program 4.20 adalah kode program dari penjelasan di atas.

```
For j = 0 To K - 1
    arr = Nothing
   If Not anakcentro(j) Is Nothing Then
   arr = anakcentro(j).Split(";")
      For g = 0 To arr.GetUpperBound(0) - 1
       Dim arr1() As String = grup(arr(g)).Anggota.Split(";")
          For i = 0 To arr1.GetUpperBound(0) - 1
           Titik(arr1(i)).Cluster = j
           Next
       Next
    End If
Next
```

Segmen Program 4.20. Penempatan Titik Inti kedalam Klaster

4.6.5 Bila Jumlah Grup Titik Inti Sama Dengan Jumlah Klaster

Sub Bab 4.6.1 hingga 4.6.4. di atas adalah implementasi jika grup titik inti lebih banyak daripada jumlah klaster (g > K). jika sama (g = K), maka langkah-langkah tersebut tidak perlu dilakukan dan hanya melakukan dua langkah, yaitu mengeset setiap titik representasi $TRep_g$ sebagai titik pusat klaster $centroid_j$ dan menempatkan titik-titik inti ke dalam $klaster_j$. Kode program dari kedua langkah ini adalah sebagai berikut:

```
For j = 0 To K - 1
    For d = 0 To centroid.GetUpperBound(1)
    centroid(j, d) = grup(j).TRep(d)
    Next
    arr = Nothing
    arr = grup(j).Anggota.Split(";")
    For i = 0 To arr.GetUpperBound(0) - 1
    Titik(arr(i)).Cluster = j
    Next
Next
```

Segmen Program 4.21. Dua Langkah Bila Jumlah Grup = Jumlah Klaster

4.7 Penentuan Akhir Titik Pusat Klaster dan Klasterisasi Titik Data

Segmen Program 4.22 adalah implementasi dari Sub Bab 3.7. Jika terdapat titik tepi (Ntepi > 0), maka klasterisasi akhir dilakukan. Terdapat dua langkah pada tahap ini yaitu pencarian klaster terdekat bagi setiap titik data dan perubahan posisi titik pusat klaster. Kedua langkah tersebut dilakukan berulang kali hingga posisi titik pusat klaster tidak berubah.

```
'hitung jarak tiap titik tepi dengan centroid
If NTepi > 0 Then
    Dim jarak, jarakMin As Single
K2:
    For i = 0 To Titik.GetUpperBound(0)
        jarakMin = Single.MaxValue
        For j = 0 To K - 1
            jarak = Euclidean1(centroid, i, j, Dimensi)
            If jarakMin > jarak Then
                jarakMin = jarak
                Titik(i).Cluster = j
            End If
        Next
        AnakCentro(Titik(i).Cluster) += i & ";"
    Next

'Geser Centroid hingga di pusat titik-titiknya
    Dim d As Integer
    Dim newCentro(K - 1,Dimensi -1) As Single
    Dim ismove As Boolean

    For j = 0 To K - 1
    arr = Nothing
       If Not anakcentro(j) Is Nothing Then
        arr = AnakCentro(j).Split(";")
          For d = 0 To centroid.GetUpperBound(1)
           For i = 0 To arr.GetUpperBound(0) - 1
               newCentro(j, d) += Titik(arr(i)).Attributes(d)
               Next
             newCentro(j, d) = newCentro(j, d)/(arr.Length - 1)
              If newCentro(j, d) <> centroid(j, d) Then
              ismove = True
                centroid(j, d) = newCentro(j, d)
              End If
           Next
        End If
    Next
    newCentro.Clear(newCentro, 0, newCentro.Length)
    'Jika Centroid pindah, maka ulangi dr langkah 2
    If ismove Then
        ismove = False
        AnakCentro.Clear(AnakCentro, 0, AnakCentro.Length)
        GoTo K2
    End If
End If
```

Segmen Program 4.22. Penentuan Akhir Titik Pusat Klaster dan Klasterisasi Titik Data

BAB 5
UJI COBA DAN EVALUASI

Pada bab ini akan dibahas tentang proses uji coba yang bertujuan untuk mengukur kinerja algoritma. Pembahasan tersebut meliputi lingkungan uji coba, data uji coba, scenario uji coba, pelaksanaan dan hasil uji coba, serta analisa hasil uji coba. Semua itu akan tersaji pada sub-bab 5.1 sampai dengan sub-bab 5.4.

5.1 Lingkungan Uji Coba

Berikut ini adalah lingkungan perangkat keras dan perangkat lunak yang digunakan untuk uji coba dalam penelitian ini :

5.1.1 Perangkat Keras

Perangkat keras yang digunakan berupa laptop dengan spesifikasi sebagai berikut :

- Processor : AMD Turion MK-38 (2.2 GHz, cache L2 512 KB)
- Memori : DDR2 512 MB
- Harddisk : 80GB

5.1.2 Perangkat Lunak

Perangkat lunak yang digunakan dalam uji coba ini adalah :

- Sistem Operasi : Microsoft Windows XP
- Bahasa Pemrograman : Basic dengan platform .Net 2003

5.2 Data Uji Coba

Pengukuran kinerja dari algoritma ini dilakukan dengan mengujikan coba algoritma ke sejumlah set data, tiga set data kecil dan lima set data besar, yaitu : Iris, Wine, New-Thyroid, SAT, Pendigit, Shuttle, Image-Seg, dan Letter. Set-set data itu diambil dari UCI Repository (http://archive.ics.uci.edu/ml/datasets.html). Tabel 5.1. menunjukkan sejumlah informasi dari masing-masing set data, yaitu jumlah data, jumlah atribut, dan jumlah klaster.

Tabel 5.1. Perincian Set Data yang Digunakan dalam Uji Coba

Set Data	Jumlah Data	Jumlah Atribut	Jumlah Klaster
Iris	150	4	3
Wine	177	13	3
New-Thyroid	215	5	3
SAT	4435	36	7
Pendigit	10992	16	10
Shuttle	14500	9	7
Image-Seg	2100	19	7
Letter	20000	16	26

Penjelasan dari masing-masing set data adalah sebagai berikut :

1. Set Data Iris (Fisher, 1936). Set data Iris adalah set data yang paling banyak digunakan untuk Datamining. Set data Iris terbagi dalam 3 klaster yang mencerminkan tiga jenis varietas bunga Iris, yaitu Iris sentosa, Iris versicolor, dan Iris Virginica, yang masing-masing terdiri dari 50 data. Setiap data memiliki 4 atribut yang menyimpan ukuran panjang sepal, lebar sepal, panjang petal, dan lebar petal. Karakteristik dari set data ini adalah terjadinya *overlapping* antara jenis Iris versicolor dan Iris virginica. Sedangkan Iris sentosa terpisah dari yang lain.
2. Set Data Wine adalah kumpulan data hasil analia kimia tentang pertumbuhan wine di suatu wilayah di Italia. Terdapat 13 atribut yang nilai-nilainya digunakan untuk membedakan data ke dalam 3 klaster yang berbeda. Klaster I beranggotakan 59 data, klaster II 70 data, dan klaster III 48 data.
3. Set Data New-Thyroid berisikan data/informasi tentang pasien thyroid yang terbagi dalam 3 tipe kelas euthyroidism, hypothyroidism, dan hyperthyroidism. Terdapat 5 atribut dari data merupakan catatan medis pasien ketika dites. Set data terdiri dari 215 data yang terdisitribusi dalam 150 euthyroidism, 35 hyperthyroidism, dan 30 hypothyroidism.
4. Set Data Sat adalah basis data sub-area 82x100 piksel dari sebuah gambar Landsat MSS dengan jumlah data 4435. Tiga puluh enam atributnya adalah empat spectral band dari setiap sembilan piksel dalam ketetanggan 3x3. Nilai-nilai atribut tersebut adalah nilai ASCII hasil konversi dari nilai binary yang dilakukan oleh Ashwin Srinivasan. Terdapat tujuh kelas pada set data ini,

yaitu tanah merah (1), ladang kapas (2), tanah keabuan (3), tanah keabuan basah (4), tanah dengan gundukan tumbuhan (5), kelas campuran (6), dan tabah keabuan yang sangat basah (7). Distribusi data tidak diketahui

5. Set Data Pendigit (1983). Set data Pendigit ini adalah kumpulan digit (0 sampai 9) yang ditulis oleh 44 orang, masing-masing menulis 250 digit secara sembarang pada kotak beresolusi 500x500 piksel. Terdapat 16 atribut yang merupakan 16 titik yang diperhatikan untuk pengenalan digit. Total data adalah 10992 dengan distribusi data masing-masing 1005 data untuk kelas 3, 5, 8, dan 9; 1056 untuk kelas 6; 1142 untuk kelas 7; 1143 untuk kelas 0 dan 1; 1144 untuk kelas 2 dan 4.
6. Set Data Shuttle yang digunakan dalam uji coba ini adalah data untuk test, yaitu sebanyak 14500, dengan 9 atribut, yang dikelompokkan dalam 7 kelas yang berbeda : Rad Flow (1), Fpv Close (2), Fpv Open (3), High (4), Bypass (5), Bpv Close (6), Bpv Open (7). Distribusi data tidak diketahui.
7. Set Data Image-Seg adalah basis data dari 7 jenis gambar pemandangan luar (*outdoor*), yaitu batu bata, langit, dedaunan, semen, jendela, jalan, dan rumput, yang masing-masing berjumlah 300 data. Gambar-gambar tersesut disimpan dalam 19 atribut kontinyu.
8. Set Data Letter adalah kumpulan 20000 data huruf (26 abjad) yang diambil dari gambar-gambar karakter dengan ukuran dan tipe huruf (*font*) yang berbeda. 16 atribut yang disimpan dalam set data ini adalah konversi dari pemindahan (*scan*) raster tiap huruf. Distribusi data untuk setiap 26 kelas seimbang.

5.3 Skenario Uji Coba

Pengukuran kinerja dari algoritma ini menerapkan beberapa skenario uji coba berikut:

1. Uji coba pengaruh nilai paramter *MinPts* terhadap kinerja algoritma.

   Algoritma penelitian ini memperoleh nilai parameter *MinPts* dari luar sistem, yaitu berupa masukan dari pengguna. Nilai *MinPts* bergantung pada karakteristik set data, dan nilainya dapat berbeda bagi satu set data dengan yang lain. Walau demikian, algoritma DBSCAN yang dikembangkan Ester,

dkk menggunakan nilai 4 sebagai *MinPts* bagi semua set data yang berdimensi dua (Ester, 1996). Namun, tidak ditemukan literatur apakah nilai *MinPts* tersebut juga optimal untuk set data yang berdimensi besar, sehingga pengguna harus mengetahui nilai *MinPts* dari set data yang hendak diklaster.

Skenario uji coba ini bertujuan untuk menemukan nilai / rentang nilai *MinPts* optimal dari sejumlah set data multi dimensi, kemudian digunakan dalam uji coba perbandingan. Nilai tersebut juga dapat diberikan sebagai saran bagi pegguna yang tidak mengetahui nilai *MinPts* sebuah set data.

2. Uji coba perbandingan algoritma penelitian ini dengan algoritma k-means berpartisi sederhana milik Hung, dkk.

   Uji coba ini akan membandingkan kualitas (kesalahan klasterisasi) dan kecepatan kinerja (waktu dalam detik) antara algoritma penelitian ini dengan algoritma k-means berpartisi sederhana milik Hung, dkk. Karena tidak tersedianya *source code* dari algoritma k-Means sederhana, maka dibuatlah program implementasi dari algoritma tersebut yang mengaju pada *pseudocode* yang terdapat di jurnal milik Hung, dkk (Hung, 2005). Sama seperti algoritma penelitian, implementasi ini juga dilakukan dalam bahasa Basic berpaltform .Net.

5.4 Pelaksanaan dan Hasil Uji Coba

Sub-bab ini akan membahas tentang pelaksanaan skenario yang dijelaskan pada sub-bab 5.3 dan juga hasil dari uji coba tersebut.

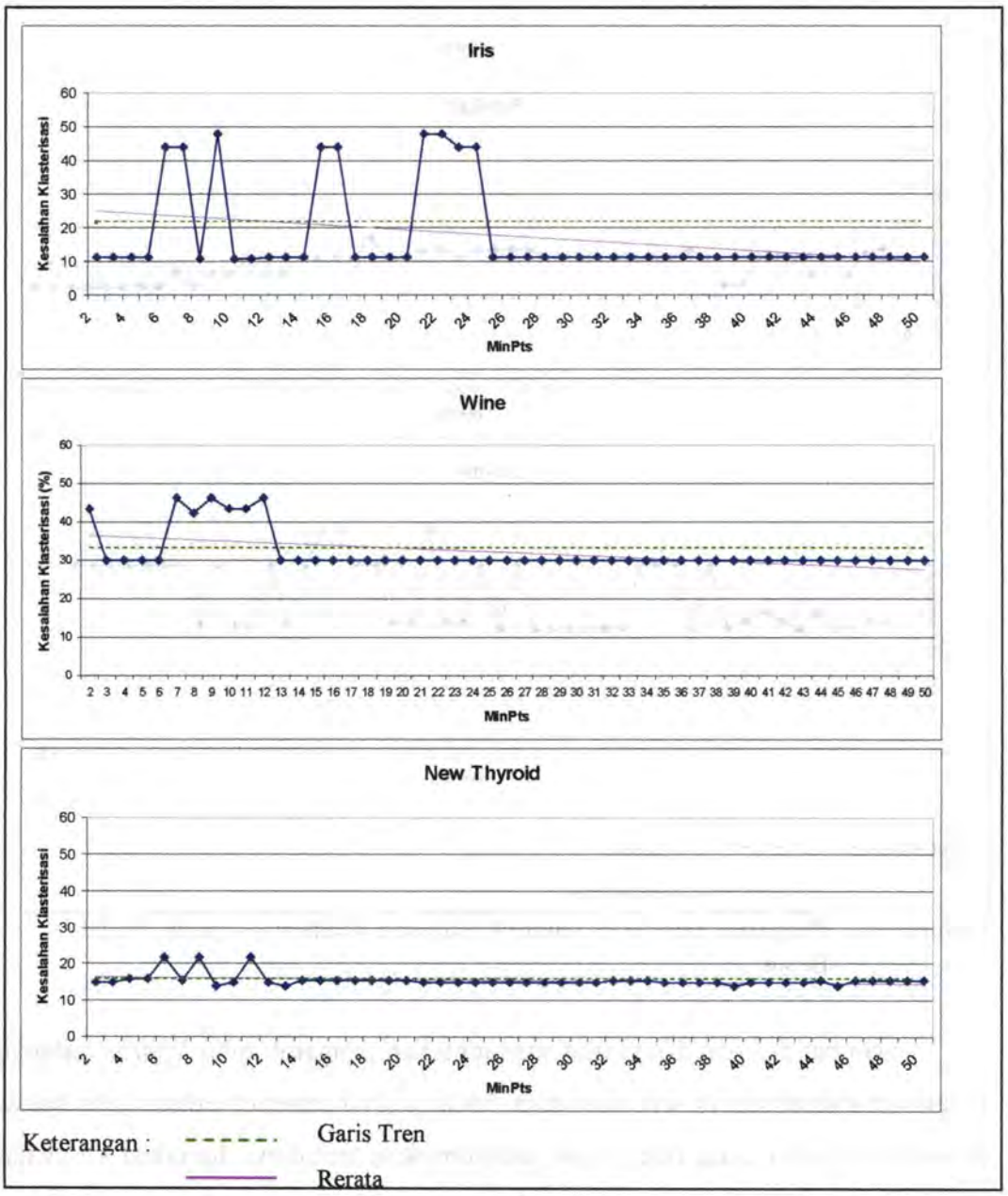
5.4.1 Uji coba Pengaruh nilai *MinPts* Terhadap Kinerja Algoritma

Awalnya uji coba pengaruh nilai *MinPts* ini dilakukan pada enam set data: Iris, Wine, New-Thyroid, SAT, Pendigit dan Shuttle. Hasilnya akan diujikan pada set data Image-Set dan letter.

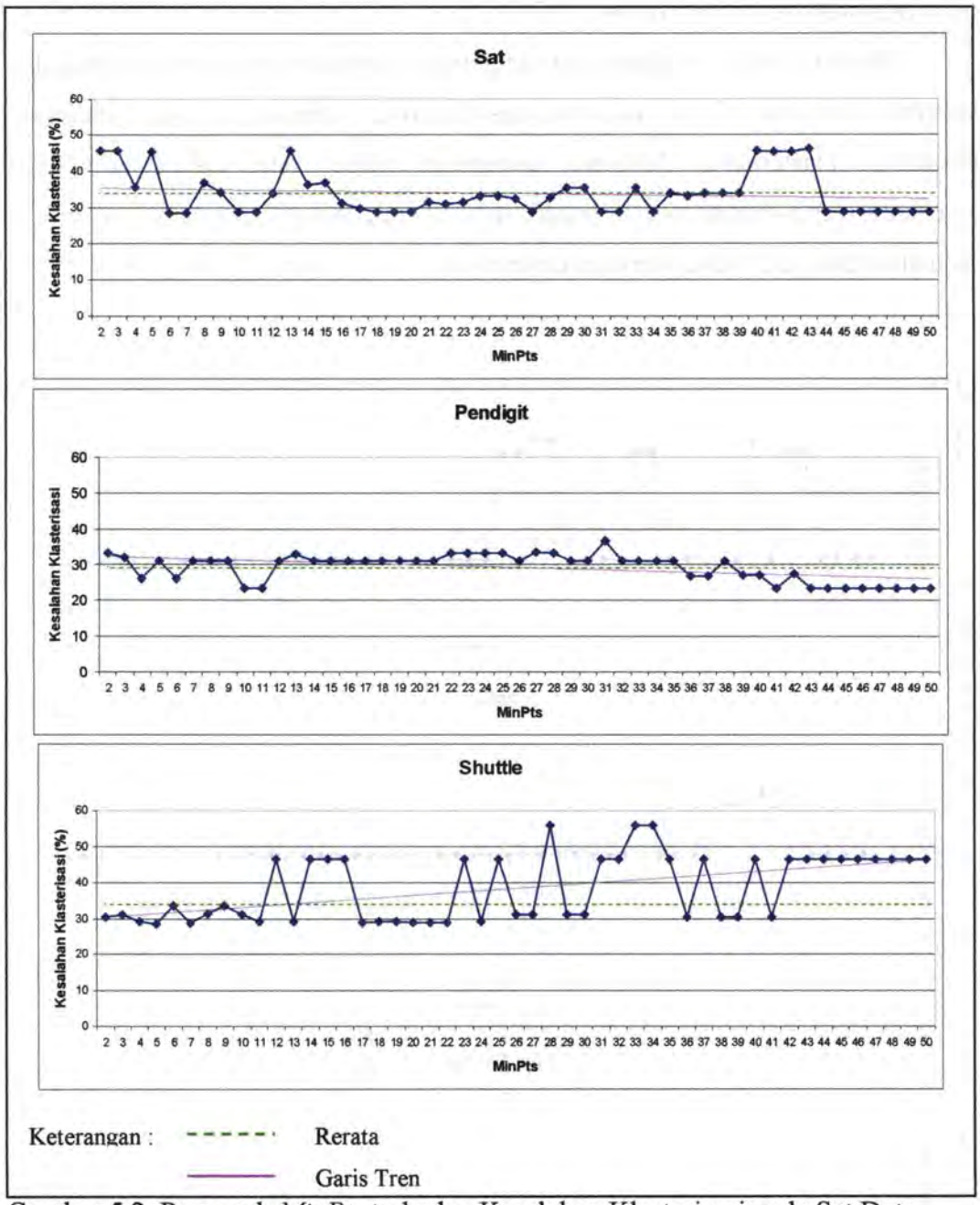
Algoritma dijalankan pada masing-masing set data dengan nilai *MinPts* antara 2 sampai dengan 50. Nilai dua adalah batas minimum *MinPts*, karena *MinPts* digunakan untuk mencari sejumlah (*MinPts* -1) tetangga terdekat dari tiap titik. Nilai 50 digunakan sebagai batas atas pengujian sebab pada nilai tersebut diduga performa akurasi dari algoritma telah stabil (tidak berbeda dengan

performa akurasi nilai *MinPts* sebelumnya) sedangkan performa kecepatan akan terus menurun (semakin lamban).

Pada uji coba ini dilakukan pengamatan terhadap performa kualitas dan kecepatan algoritma. Performa kualitas diperoleh dengan melihat persentase kesalahan klasterisasi. Performa kecepatan dilihat dari waktu komputasi algoritma, yang didalamnya termasuk juga banyak iterasi yang harus dilakukan oleh algoritma pada setiap set data dengan nilai *MinPts* antara 2 hingga 50.



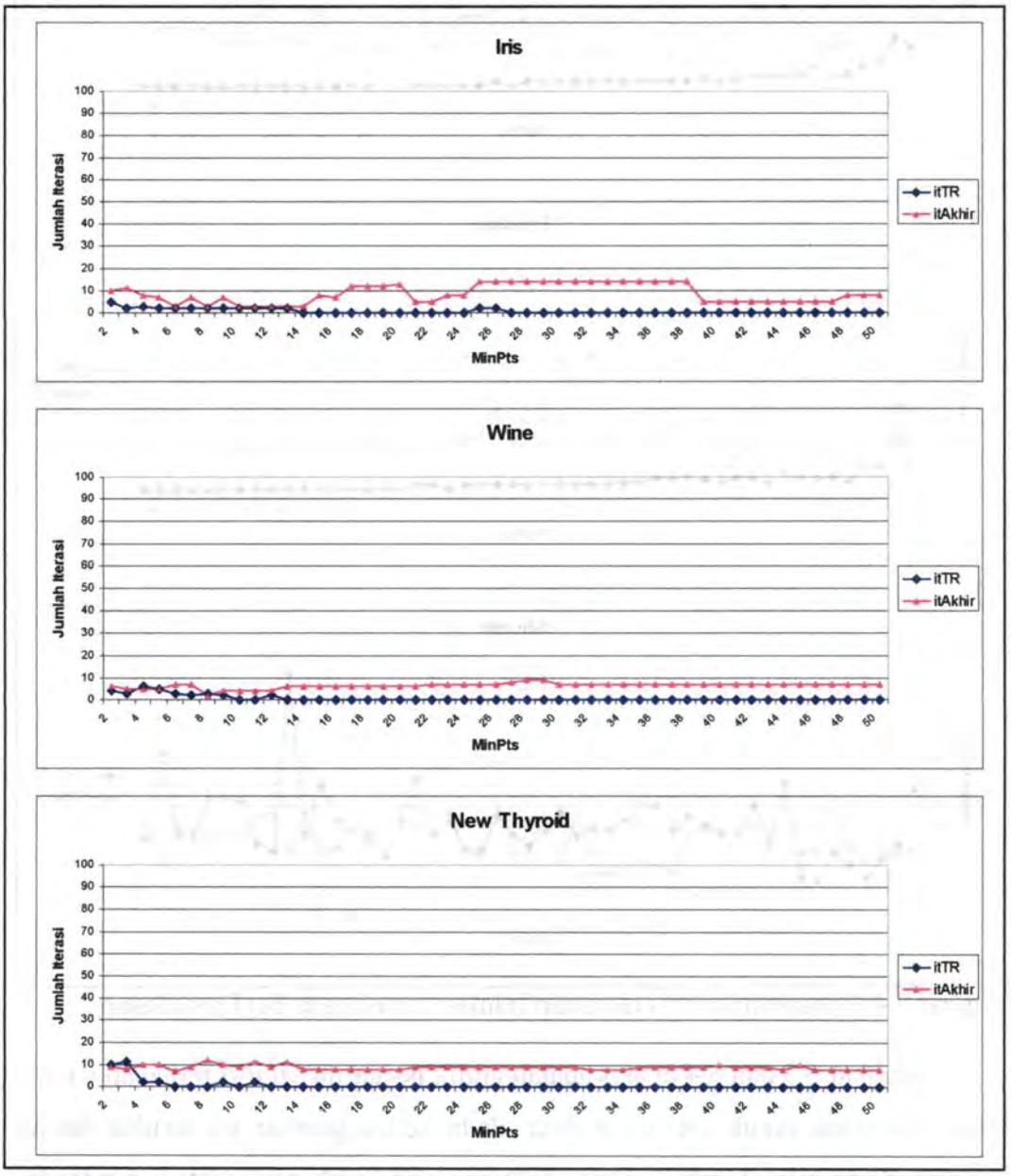
Gambar 5.1. Pengaruh *MinPts* terhadap Kesalahan Klasterisasi pada Set Data Kecil



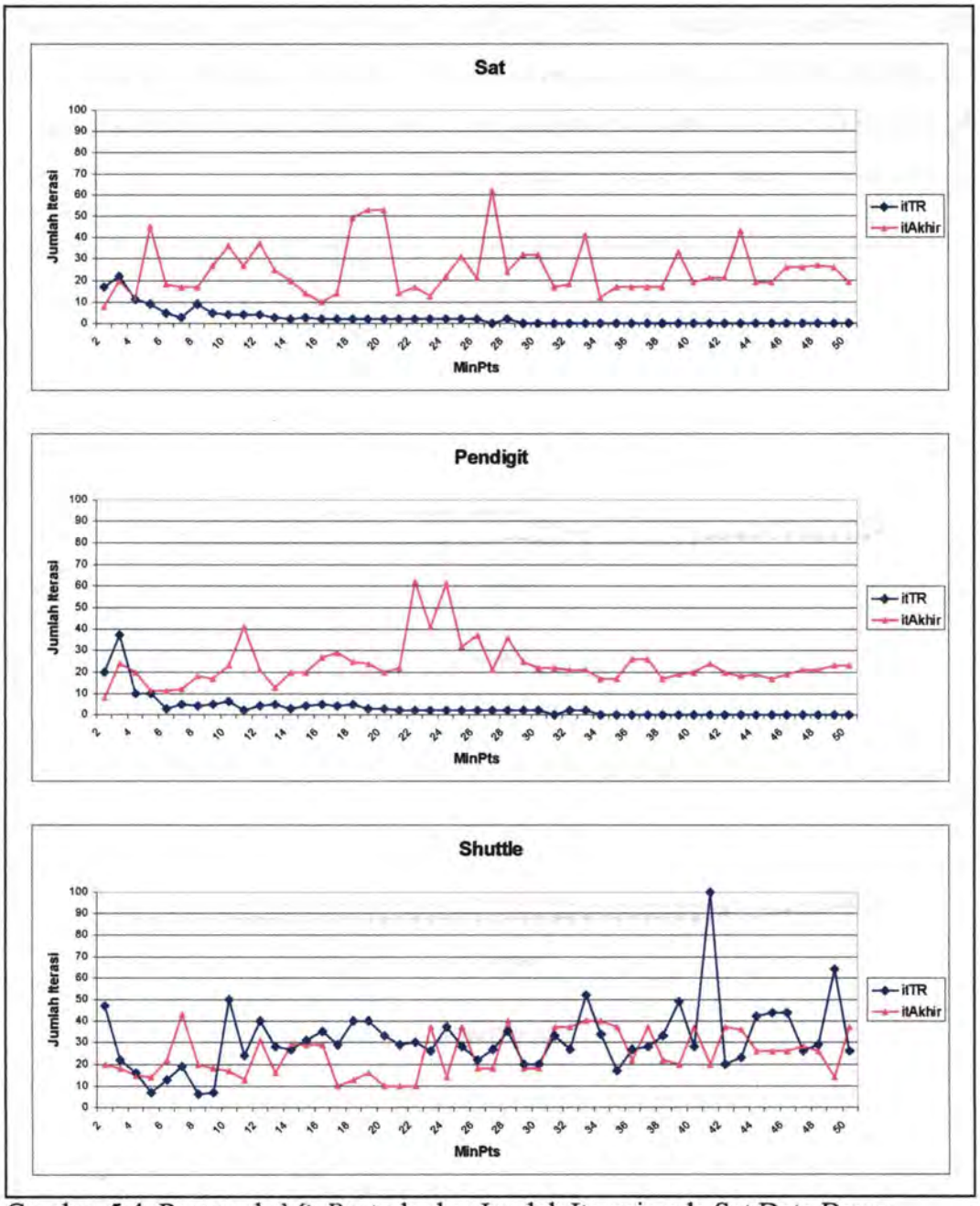
Gambar 5.2. Pengaruh *MinPts* terhadap Kesalahan Klasterisasi pada Set Data Besar

Gambar 5.1 dan 5.2 di atas menunjukkan pengaruh nilai *MinPts* terhadap kesalahan klasterisasi di setiap set data. Pada gambar tersebut terlihat jelas bahwa pemilihan *MinPts* yang tidak tepat menyebabkan terjadinya lonjakan kesalahan klasterisasi, hal ini terutama terlihat jelas pada grafik untuk set data Iris dan Wine. Meski garis tren di tiap grafik menurun seiring dengan bertambahnya nilai

MinPts, namun sebenarnya tidak terdapat hubungan linear antara kesalahan klasterisasi dengan pertambahan nilai *MinPts*. Bahkan seluruh set data yang digunakan dalam uji coba ini mencapai hasil stabil (tidak lagi menurun kesalahan klasterisasinya) pada saat *MinPts* kurang dari 50.

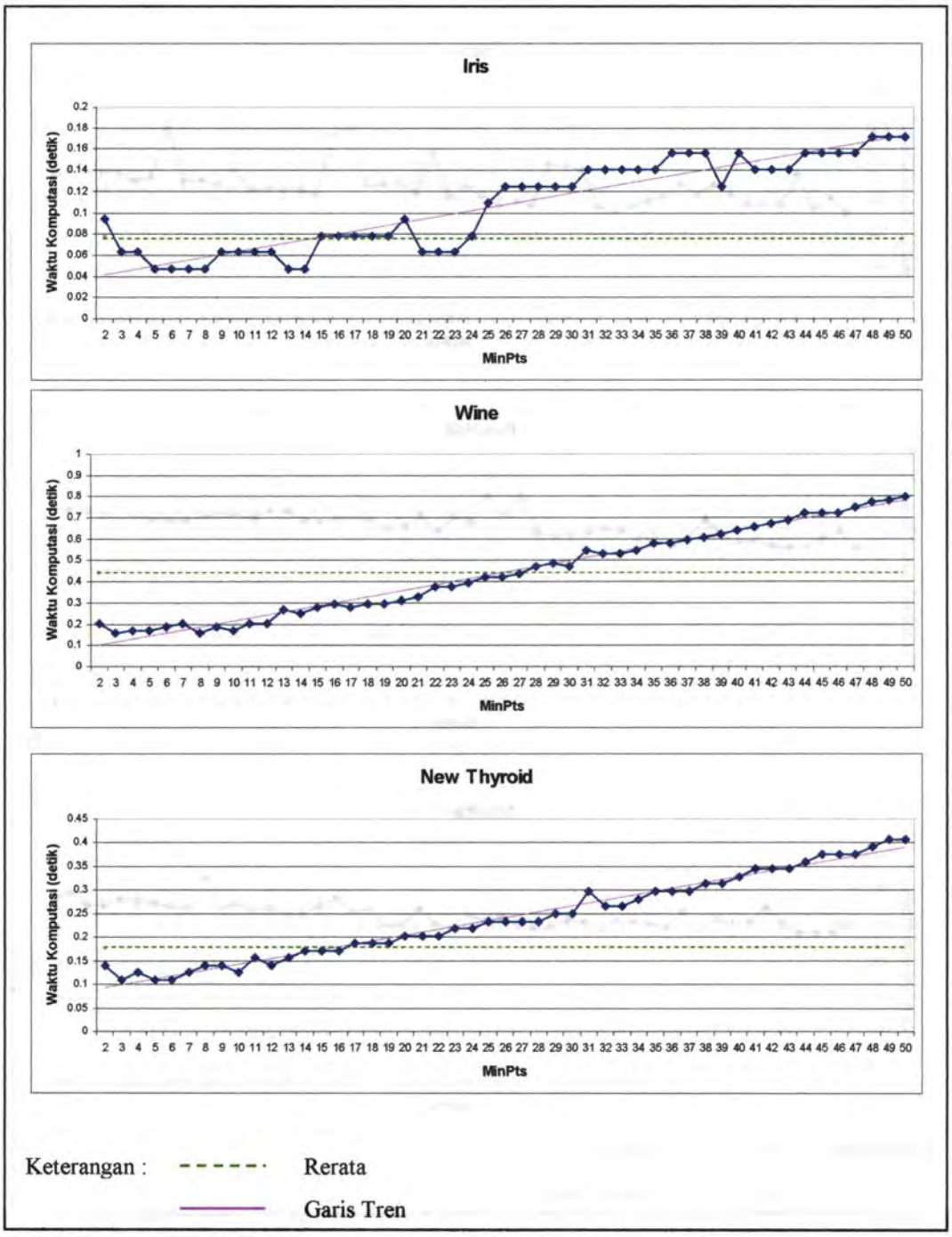


Gambar 5.3. Pengaruh *MinPts* terhadap Jumlah Iterasi pada Set Data Kecil

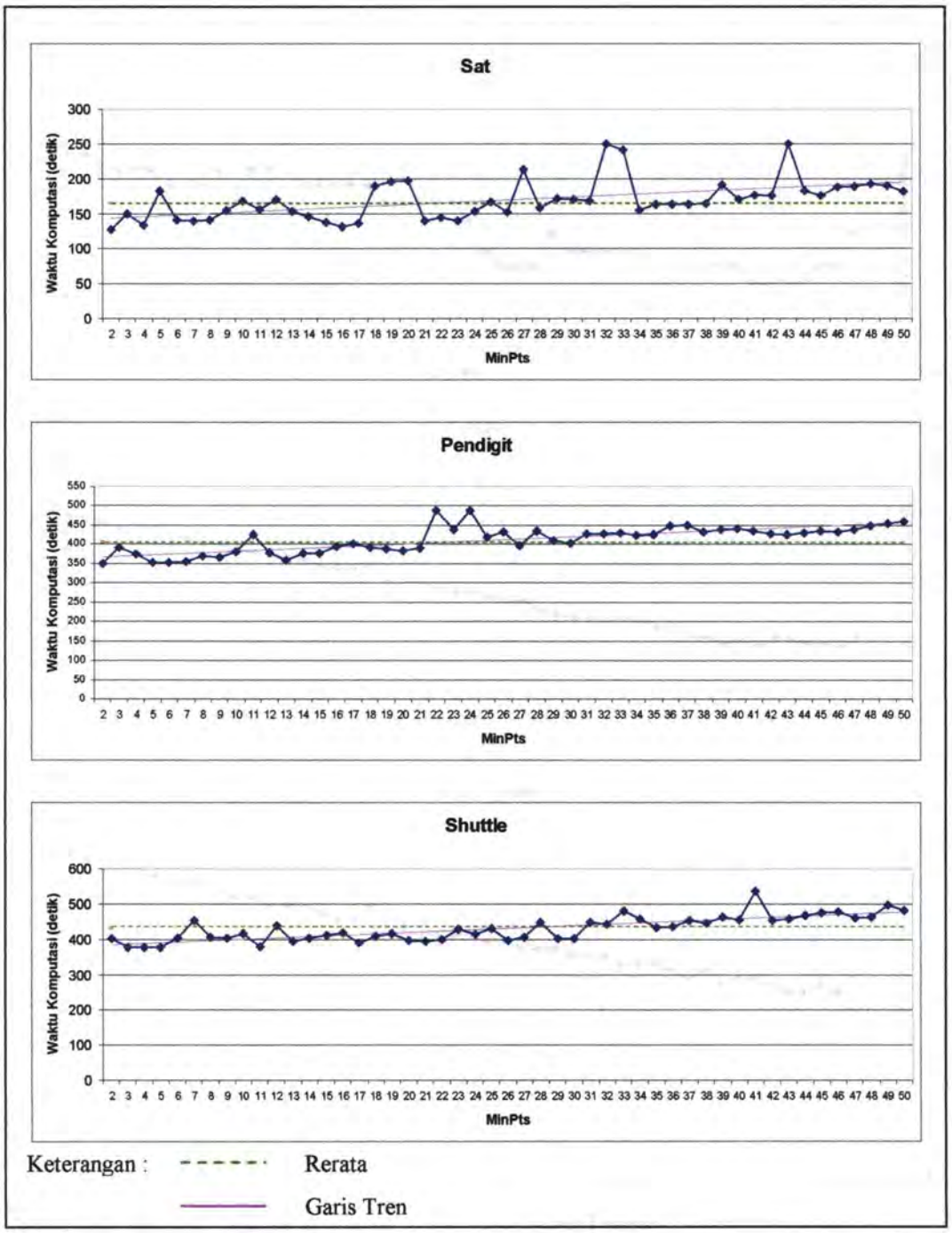


Gambar 5.4. Pengaruh *MinPts* terhadap Jumlah Iterasi pada Set Data Besar

Gambar 5.3 dan 5.4 di atas adalah grafik pengaruh *MinPts* terhadap jumlah iterasi algoritma untuk setiap set data. Pada kedua gambar itu terlihat bahwa semakin besar nilai *MinPts*, iterasi k-Means pada titik representasi cenderung semakin sedikit, bahkan hingga 0 atau tidak perlu melakukan k-Means pada titik representasi. Namun, apabila k-Means pada titik representasi tidak dilakukan, iterasi pada k-Means akhir akan terjadi berulang kali. Sehingga perlu dilakukan pemilihan *MinPts* yang tepat agar dapat menghasilkan algoritma yang efisien.



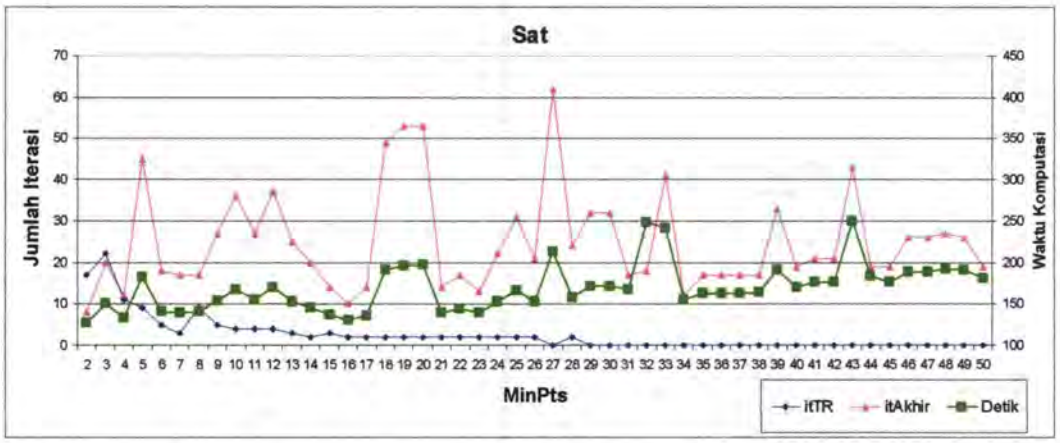
Gambar 5.5. Pengaruh *MinPts* terhadap Waktu Komputasi pada Set Data Kecil



Gambar 5.6. Pengaruh *MinPts* terhadap Waktu Komputasi pada Set Data Besar

Seperti terlihat pada Gambar 5.5 dan 5.6, secara umum tren waktu komputasi terhadap nilai *MinPts* adalah linear, artinya semakin besar nilai *MinPts*, semakin lama waktu komputasi. Namun waktu komputasi juga dipengaruhi oleh banyaknya iterasi. Hal ini terlihat jelas pada set-set data besar, terutama pada set data Sat yang waktu komputasinya cukup fluktuatif. Ternyata pola grafik tersebut

setara dengan pola grafik jumlah iterasinya, seperti yang diperlihatkan Gambar 5.7 berikut :

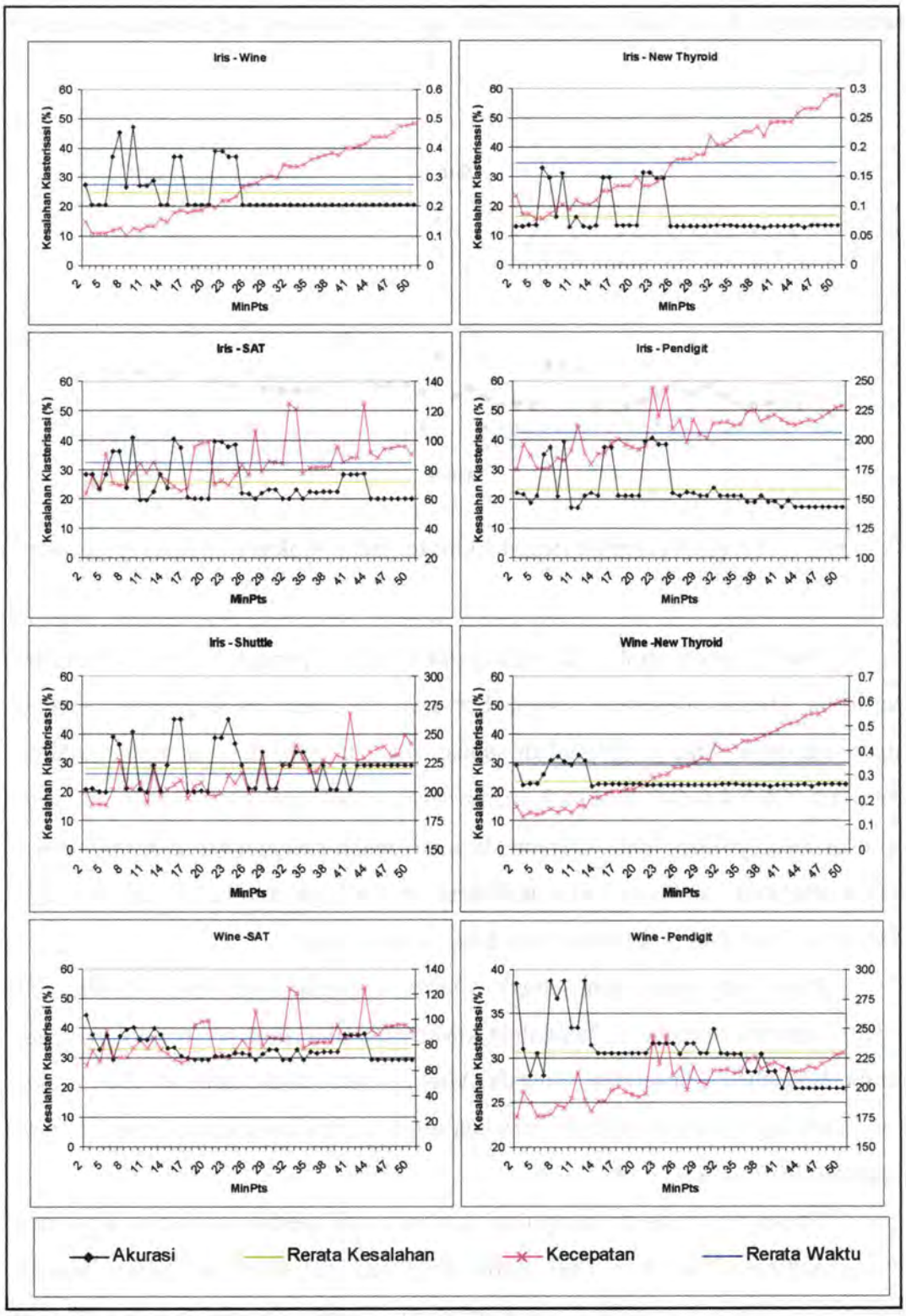


Gambar 5.7. Pengaruh Jumlah Iterasi Akhir terhadap Waktu Komputasi pada Set Data Sat.

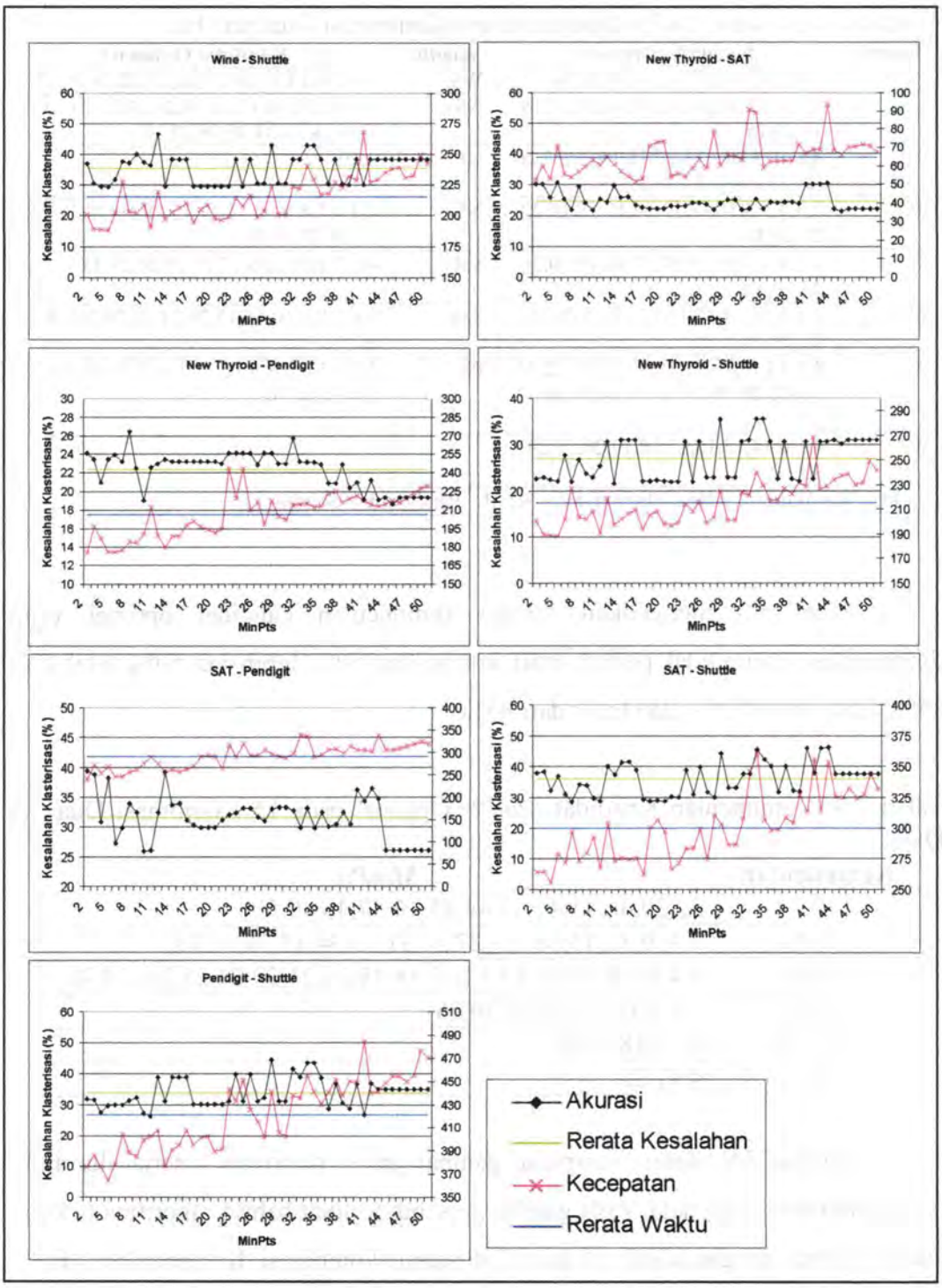
Dari gambar-gambar di atas terlihat bahwa pengaruh *MinPts* terhadap performa kinerja algoritma sangat bervariasi. Dengan nilai *MinPts* tertentu, algoritma dapat bekerja optimal di sebuah set data, tapi bekerja buruk pada set data lain. Oleh karena itu, untuk memperoleh nilai / rentang nilai *MinPts* yang optimal bagi, paling tidak, keenam data ini, maka pengamatan dengan melihat rerata pengaruh *MinPts* terhadap performa kinerja algoritma pada tiap kombinasi 2 set data, 3 set data, 4 set data, 5 set data, dan 6 set data.

Pada pengamatan ini sebuah *MinPts* dianggap layak menjadi kandidat *MinPts* optimal apabila: a) kesalahan klasterisasi algoritma lebih kecil daripada rerata kesalahan klasterisasi seluruh *MinPts* pada tiap kombinasi, b) waktu komputasi algoritma lebih cepat daripada rerata waktu komputasi seluruh *MinPts* pada tiap kombinasi.

Gambar 5.8 adalah kumpulan gambar grafik performa kinerja algoritma pada kombinasi 2 set data. Dari grafik-grafik tersebut diketahui bahwa terdapat MinPts yang muncul hampir di tiap kombinasi sebagai kandidat optimal, ada juga yang tidak pernah muncul. Tabel 5.2 menampilkan kandidat optimal di masing-masing kombinasi.



Gambar 5.8(a) Rerata Peforma Kinerja Algoritma pada Dua Set Data



Gambar 5.8(b) Rerata Peforma Kinerja Algoritma pada Dua Set Data (lanjutan)

Tabel 5.2. Kandidat *MinPts* Optimal dalam Kombinasi Dua Set Data

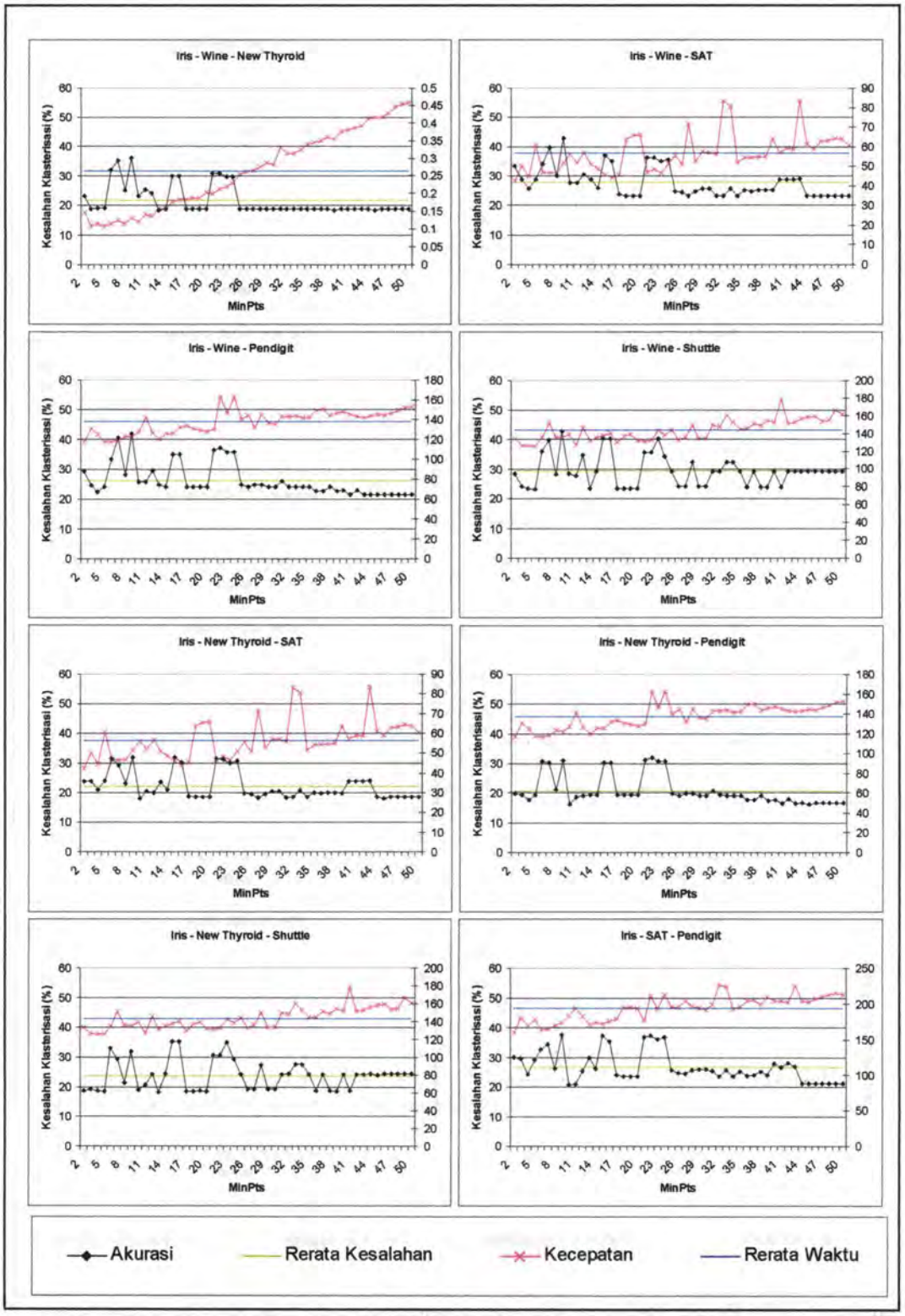
Komb.	Kandidat Optimal	Komb.	Kandidat Optimal
IW	3,4,5,13,14,17,18,19,20,25,26	WS	3,4,5,6,13,17,18,19,20,21,22,24,26,27
IN	2,3,4,5,8,10,11,12,13,14,15,17,18,19,20,25	NSa	7,9,10,12,16,17,18,19,20,21,22,23,24,25,26,28,31,34,35,36,37,38
ISa	4,8,10,11,12,14,17,18,19,20,25,26,28,30,31,34,35,36,37,38	NP	4,10,
IP	2,3,4,5,8,10,12,13,14,17,18,19,20,27,29,30	NS	2,3,4,5,8,9,10,11,13,17,18,19,20,21,22,24,26,27,29,30
IS	2,3,4,5,8,10,11,13,17,18,19,20,26,27,29,30	SaP	4,6,7,10,11,16,17,18,19,20,21,33
WN	3,4,5,13,14,15,16,17,18,19,20,21,22,23,24,25,26	SaS	4,6,7,8,9,10,11,17,20,21,22,24,26,29,30
WSa	4,6,14,15,16,17,18,19,20,21,22,23,24,25,26,28,29,30,31,34,35,36,37,38	PS	2,3,4,5,6,7,8,9,10,11,13,17,18,19,20,21,26,27,29,30
WP	4,5,6,14,15,16,17,18,19,20,21,29,30		
I = Iris, W = Wine, N = New Thyroid, Sa = Sat, P = Pendigit, S = Shuttle			

Tabel 5.3 menyajikan jumlah kemunculan kandidat optimal yang dikriteriakan dalam tidak pernah (nol), kurang dari 50%, lebih dari 50%, lebih dari 75%, lebih dari 87,5%, dan lebih dari 95%.

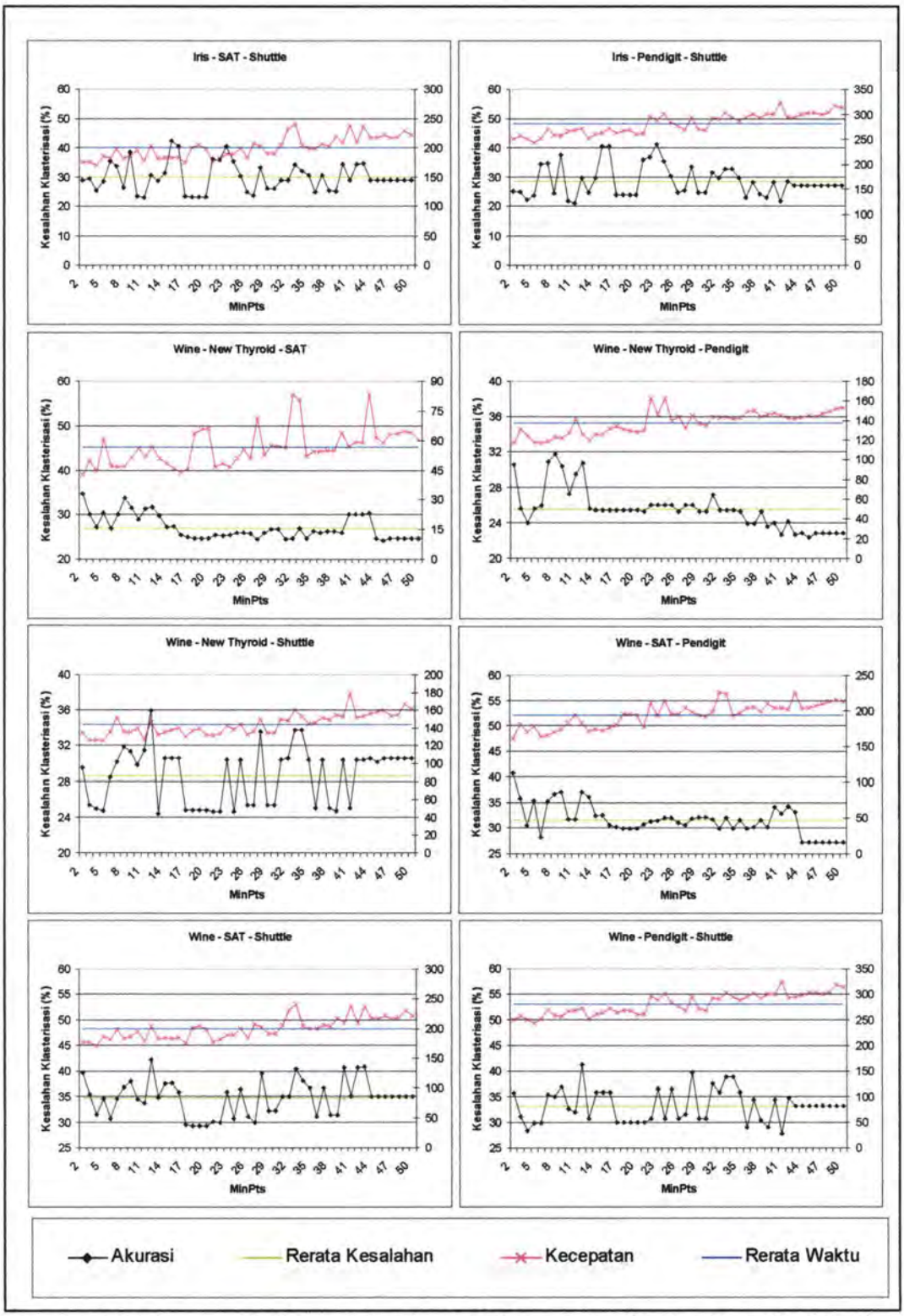
Tabel 5.3. Kemunculan Kandidat *MinPts* Optimal pada 15 Kombinasi Dua Set Data

Kemunculan	*MinPts*
0	32,39,40,41,42,43,44,45,46,47,48,49,50
< 50%	2,7, 9,12,15,16, 23,27,28,31,33,34,35,36,37,38
≥ 50%	3,4,5,6,8,10,11,13,14,17,18,19,20,21,22,24,25,26, 29,30
≥ 75%	3,4,5,10,17,18,19,20,26
≥ 87,5%	4,17,18,19,20
≥ 95%	4,17,20

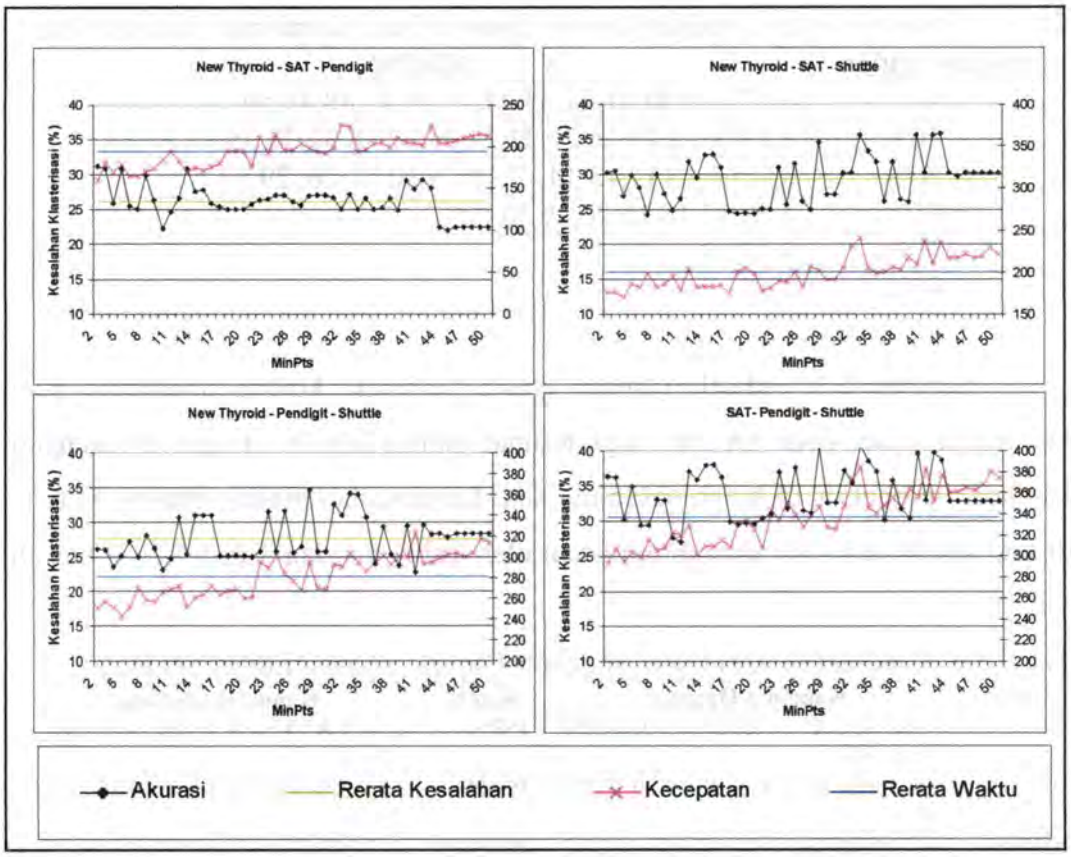
Gambar 5.9 adalah kumpulan gambar grafik performa kinerja algoritma pada kombinasi 3 set data. Pada gambar tersebut terlihat bahwa algoritma bekerja cukup optimal dengan MinPts 4 dan 17 di semua kombinasi. Kemunculan MinPts sebagai kandidat optimal pada 20 kombinasi tersebut ditunjukkan oleh Tabel 5.4, sedangkan frekuensi kemuculan MinPts sebagai kandidat optimal disajikan pada Tabel 5.5.



Gambar 5.9(a) Rerata Peforma Kinerja Algoritma pada Tiga Set Data



Gambar 5.9(b) Rerata Peforma Kinerja Algoritma pada Tiga Set Data (lanjutan)



Gambar 5.9(c) Rerata Peforma Kinerja Algoritma pada Tiga Set Data (lanjutan)

Tabel 5.4. Kandidat *MinPts* Optimal dalam Tiap Kombinasi Tiga Set Data

Komb.	Kandidat Optimal	Komb.	Kandidat Optimal
IWN	3,4,5,13,14,17,18,19,20,25,26	WNSA	4,6,14,15,16,17,18,19,20,21,22,23,24,25,26,28,29,30,31,34,35,36,37,38
IWSA	4,10,11,14,17,25,26,28,29,30,31,34,35,36,37,38	WNP	3,4,5,13,14,15,16,17,18,19,20,21,29,30
IWP	3,4,5,10,11,13,14,17,18,19,20,27,29,30	WNS	3,4,5,6,13,17,18,19,20,21,22,24,26,27,29,30,36
IWS	2,3,4,5,8,10,11,13,17,20,26,27,29,30,36	WSAP	4,6,16,17,18,19,20,21,23,26
INSA	4,10,11,13,17,25,26,28,29,30,34,35,36,37,38	WSAS	4,5,6,7,10,11,17,18,21,22,24,26,29,30
INP	2,3,4,5,10,12,13,14,17,18,19,20,27,29,30	WPS	3,4,5,6,10,11,13,17,18,19,20,21,26,27
INS	2,3,4,5,8,10,11,12,13,14,17,18,19,20,26,27,29,30	NSAP	4,6,7,10,11,16,17,18,19,20,21
ISAP	4,8,10,11,12,14,17,18,19,20,25,2629,30,34	NSAS	4,6,7,9,10,11,16,17,18,20,21,22,23,25,28,29
ISAS	2,3,4,5,8,10,11,13,17,18,20,26,29,30,36	NPS	2,3,4,5,6,7,9,10,11,13,17,18,19,20,21,27,29,30
IPS	2,3,4,5,8,10,11,12,13,14,17,18,19,20,26,2729,30	SAPS	4,6,7,8,9,10,11,17,18,19,20,21,29,30
I = Iris, W = Wine, N = New Thyroid, Sa = Sat, P = Pendigit, S = Shuttle			

Tabel 5.5. Kemunculan Kandidat *MinPts* Optimal Pada Kombinasi Tiga Set Data

Kemunculan	***MinPts***
0	32,33,39,40,41,42,43,44,45,46,47,48,49,50
< 50%	2,7,8,9,12,15,16,22,23,24,25,27,28,31,34,35,36,37,38
≥ 50%	3,4,5,6,10,11,13,14,17,18,19,20,21,26, 29,30
≥ 75%	4,10,17,18,19,20,29,30
≥ 87,5%	4,17,18,
≥ 95%	4,17

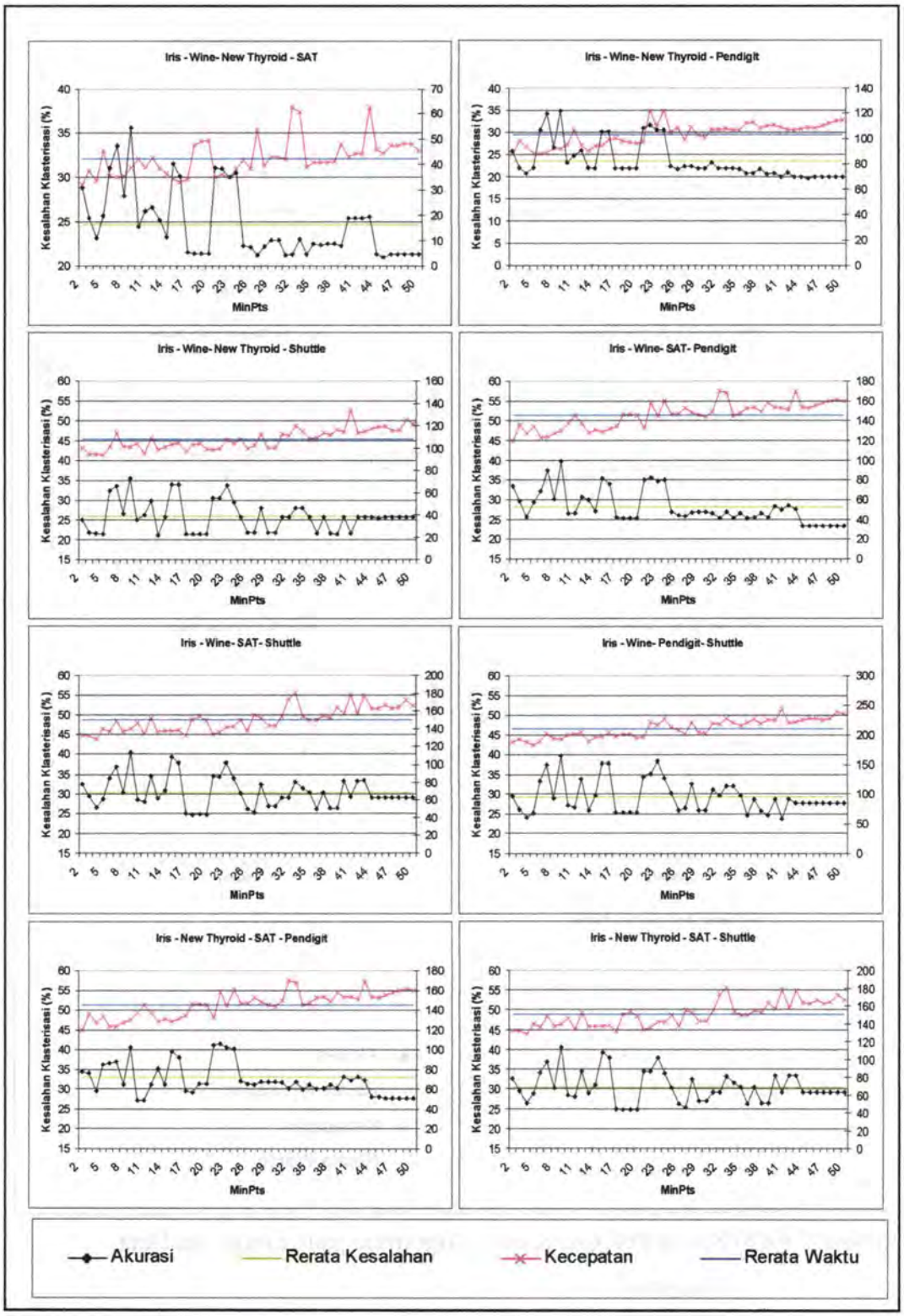
Gambar 5.10 adalah gambar grafik performa kinerja algoritma pada kombiansi 4 set data. Di sini juga terlihat bahwa MinPts 4 dan 17 menjadi kandidat optimal di 15 kombinasi yang ada. Kemunculan masing-masing MinPts di tiap kombinasi 4 set data ditunjukkan oleh Tabel 5.6 dan Tabel 5.7. di bawah ini.

Tabel 5.6. Kandidat *MinPts* Optimal dalam Tiap Kombinasi Empat Set Data

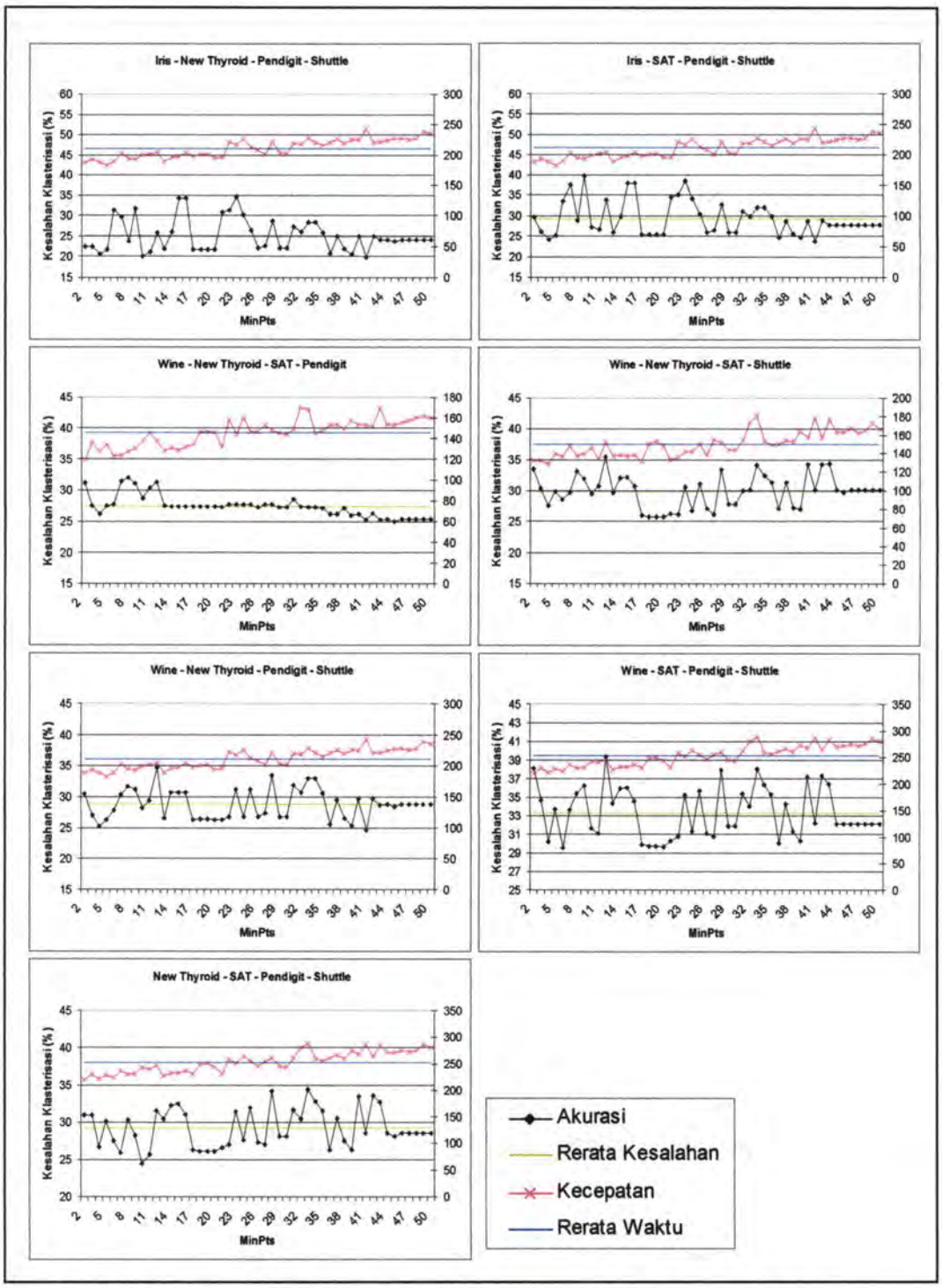
Komb.	**Kandidat Optimal**	**Komb.**	**Kandidat Optimal**
IWNSA	4,10,14,17,25,26,28,29,30,31,34,35, 36,37,38	INPS	2,3,4,5,8,10,11,13,17,18,19,20,26,27 ,29,30
IWNP	3,4,5,10,11,13,14,17,18,19,20,27,29, 30	ISAPS	2,3,4,5,8,10,11,13,17,18,19,20,26,27 ,29,30
IWNS	2,3,4,5,10,13,17,18,19,20,25,26,27,2 9,30	WNSAP	4,13,14,15,16,17,18,19,20,21,26,29, 30
IWSAP	4,10,11,14,17,18,19,20,25,26,29,30, 34	WNSAS	4,6,10,13,17,18,20,21,22,24,26,29 30
IWSAS	3,4,5,10,11,13,17,18,20,25,26,29,30,	WNPS	3,4,5,6,10,13,17,18,19,20,21,26,27,2 9,30
IWPS	3,4,5,8,10,11,13,17,18,19,20,26,27,2 9,30	WSAPS	4,6,10,1117,18,19,20,21,26,29,30
INSAP	4,8,10,11,12,14,17,18,19,20,36	NSAPS	4,6,7,9,10,11.17,18,19,20,29,30
INSAS	3,4,5,10,11,13,17,18,19,20,25,26,27, 29,20,36		
I = Iris, W = Wine, N = New Thyroid, Sa = Sat, P = Pendigit, S = Shuttle			

Tabel 5.7. Kemunculan Kandidat *MinPts* Optimal Pada Kombinasi Empat Set Data

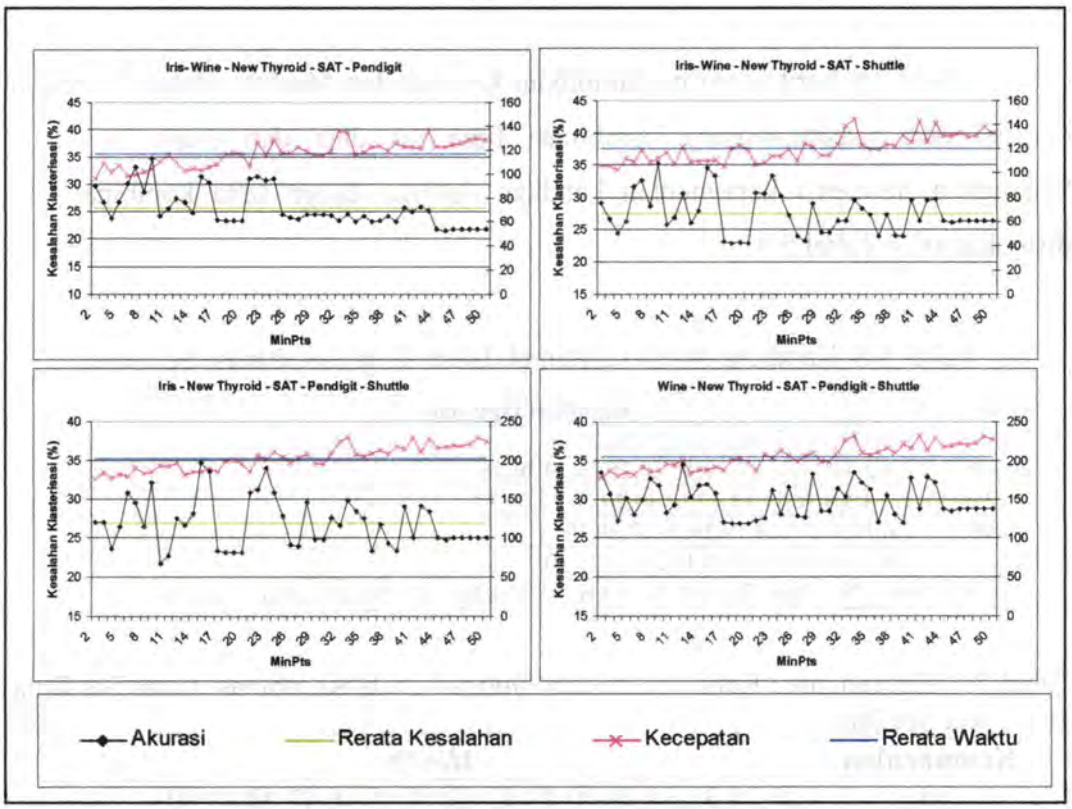
Kemunculan	***MinPts***
0	23,24,32,33,39,40,41,42,43,44,45,46,47,48,49,50
< 50%	2,6,7,8,9,12,14,15,16,21,22,25,28,31,34,35,36,37,38
≥ 50%	3,4,5,10,11,13,17,18,19,20,26,27,29,30
≥ 75%	4,10,17,18,19,20,26,29,30
≥ 87,5%	4,10,17,18,20,29,30
≥ 95%	4,17



Gambar 5.10(a) Rerata Peforma Kinerja Algoritma pada Empat Set Data

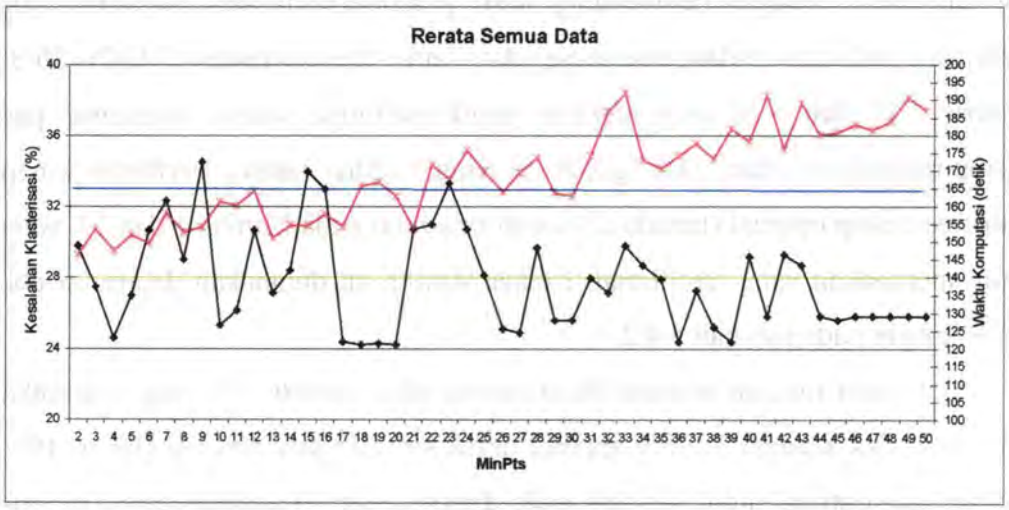


Gambar 5.10(b) Rerata Peforma Kinerja Algoritma pada Empat Set Data (lanjutan)



Gambar 5.11 Rerata Peforma Kinerja Algoritma pada Lima Set Data

Gambar 5.11 di atas adalah gambar grafik performa kinerja algoritma pada kombinasi lima set data. Dalam keempat kombinasi tersebut, lagi-lagi, MinPts 4 dan 17 selalu menjadi kandidat optimalnya. Demikian pula pada kombinasi enam set data (Gambar 5.12), algoritma bekerja optimal dengan kedua MinPts itu.



Gambar 5.12 Rerata Performa Kinerja Algoritma pada Semua Data

Tabel 5.8 berikut ini menampilkan kemunculan MinPts sebagai kandidat optimal dari masing-masing kombinasi lima set data dan enam set data. Sedangkan frekuensi kemunculan kandidat optimal dalam lima kombinasi itu disajukan oleh Tabel 5.9.

Tabel 5.8. Kandidat *MinPts* Optimal dalam Tiap Kombinasi Set Data

Komb.	Kandidat Optimal
IWNSAP	4,10,11,14,17,18,19,20,25,26,29,30
IWNSAS	3,4,5,10,11,13,17,18,19,20,25,26,29,30,36
INSAPS	2,3,4,5,8,10,11,13,17,18,19,20,26,27,29,30
WNSAPS	4,6,10,11,17,18,19,20,26,27,29,30
ALL	3,4,5,10,11,13,17,18,19,20,26,29,30
I = Iris, W = Wine, N = New Thyroid, Sa = Sat, P = Pendigit, S = Shuttle, ALL = Semua	

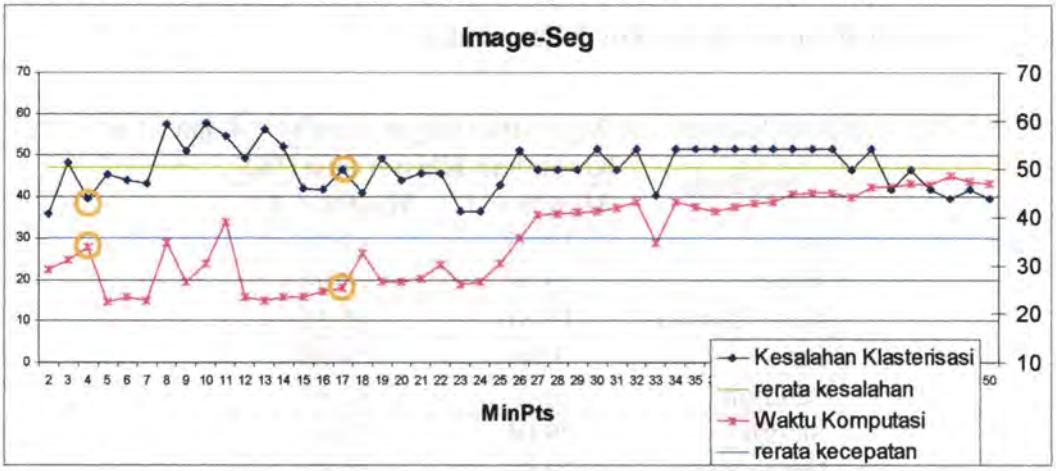
Tabel 5.9. Kemunculan Kandidat *MinPts* Optimal Pada Kombinasi Lima Set Data dan Enam Set Data

Kemunculan	*MinPts*
0	23,24,32,33,39,40,41,42,43,44,45,46,47,48,49,50
< 50%	2,6,7,8,9,12,14,15,16,21,22,25,28,31,34,35,36,37,38
≥ 50%	3,4,5,10,11,13,17,18,19,20,26,27,29,30
≥ 75%	4,10,17,18,19,20,26,29,30
≥ 87,5%	4,10,17,18,20,29,30
≥ 95%	4,17

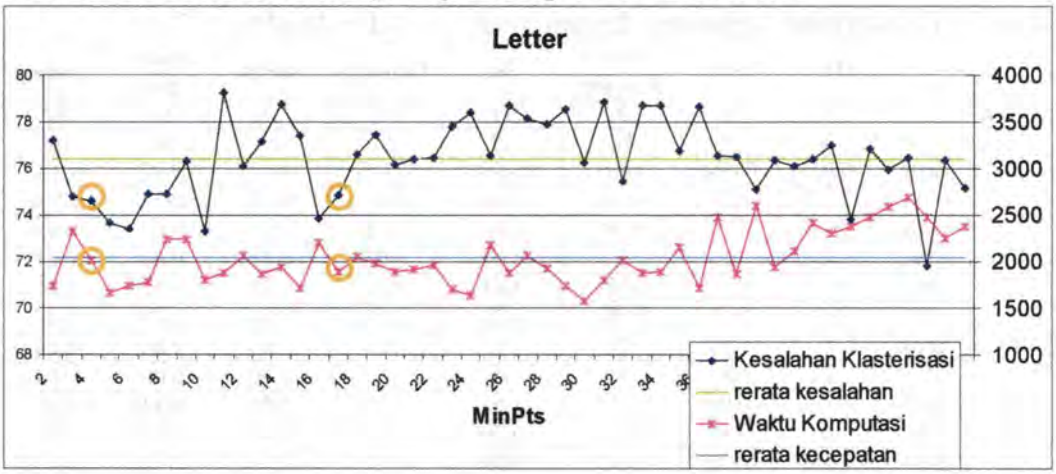
Dari percobaan diatas terlihat bahwa masing-masing *MinPts* 4 dan 17 menjadi kandidat optimal dalam 53 kombinasi dari 54 kombinasi set data yang diujikan coba. Sebagai pembanding hasil percobaan di atas, algoritma juga dijalankan pada dua set data, Image-Seg dan Letter, dengan rentang MinPts 20-50. Gambar 5.13 dan 5.14 menunjukkan grafik performa kinerja algoritma pada masing-masing set data. Dari grafik tersebut terlihat bahwa performa kinerja algoritma cukup optimal (berada di bawah rata-rata) pada MinPts 4 dan 17, walau bukan merupakan yang teroptimal. Kedua MinPts ini digunakan dalam uji coba perbandingan pada sub-bab 5.4.2.

Uji coba ini pun membuktikan bahwa nilai *MinPts* = 4 yang disarankan oleh Ester, dkk sebagai *MinPts* optimal untuk set data dua dimensi (Ester, 1996) ternyata juga optimal untuk set data multi dimensi yang digunakan dalam uji coba ini.

Gambar 5.13 Performa Kinerja Algoritma pada Set Data Image-Seg



Gambar 5.14 Performa Kinerja Algoritma pada Set Data Letter



5.4.2 Uji Coba Perbandingan Kinerja Algoritma dengan Algoritma k-means berpartisi Sederhana

Uji coba perbandingan bertujuan untuk mengukur kinerja dari algoritma penelitian ini, baik dari aspek kualitas dan kecepatan kinerja. Sama seperti pada uji coba di sub-bab 5.4.1, kualitas dari masing-masing algoritma akan diukur berdasarkan persetase kesalahan klasterisasi sedangkan kecepatan kinerja diukur berdasarkan waktu komputasi. Set data yang digunakan dalam pengujian adalah delapan set data yang telah dibahas pada sub-bab 5.2.

Algoritma yang dikembangkan ini dijalankan dengan menggunakan nilai *MinPts* hasil uji coba pada sub-bab 5.4.1, yaitu 4 dan 17. Tabel 5.10 menampilkan kesalahan klasterisasi yang dilakukan algoritma dengan kedua *MinPts* tersebut.

Sedangkan Tabel 5.11 menampilkan performa kecepatan kinerja algoritma secara detil, yaitu jumlah iterasi dan waktu di tiap modul.

Tabel 5.10. Kesalahan Klasterisasi Algoritma dengan *MinPts* = 4 dan *MinPts* = 17

Set Data	Kesalahan Klasterisasi (%)	
	***MinPts* = 4**	***MinPts* = 17**
Iris	11.33	11.33
Wine	29.94	29.94
New Thyroid	15.81	15.35
Sat	35.6	29.65
Pendigit	26.1	31.07
Shuttle	29.04	29.0
Image Seg	39.62	46.62
Letter	74.58	74.84

Tabel 5.11 Kecepatan Algoritma dengan *MinPts* = 4 dan *MinPts* = 17

			Iris	Wine	New Thyroid	Sat	Pendigit	Shuttle	Image Set	Letter
MinPts = 4	itTR		3	6	2	11	10	16	11	82
	itAkhir		8	5	10	12	20	15	44	100
	T1	detik	0.031	0.094	0.063	114.64	323.72	343.17	16.63	962.85
		%	33.33	46.3	40.38	84.65	86.5	90.31	48.06	47.90
	T2	detik	0.00	0.00	0.00	0.03	0.11	0.2	0.02	0.13
		%	0	0	0	0.02	0.03	0.05	0.05	0.01
	T3	detik	0.031	0.047	0.031	1.67	10.81	2.11	0.52	2.81
		%	33.33	23.15	19.87	1.23	2.9	0.56	1.49	0.14
	T4	detik	0	0	0	0	0	0	0.00	0.00
		%	0	0	0	0	0	0.00	0.00	0.00
	T5	detik	0.00	0.016	0.016	1.03	1.28	1.86	0.34	96.83
		%	0	7.9	10.26	0.76	0.34	0.49	0.99	4.82
	T6	detik	0.031	0.047	0.047	18.06	47.33	32.64	17.09	947.37
		%	33.33	23.15	30.13	13.34	12.64	8.59	49.41	47.13
	Tot	detik	0.094	0.203	0.156	135.44	374.25	379.98	34.59	2010
MinPts = 17	itTR		0	0	0	2	4	29	0	8
	itAkhir		12	6	8	14	29	10	17	75
	T1	detik	0.047	0.188	0.094	117.66	329.61	325.08	18.38	1090.52
		%	50	63.30	50.00	83.72	82.15	83.39	71.84	57.58
	T2	detik	0.00	0	0.016	0.13	0.33	1.14	0.05	8.61
		%	0	0.00	8.51	0.09	0.08	0.29	0.18	0.45
	T3	detik	0.00	0.016	0.016	1.75	2.02	2.38	0.53	3.52
		%	0	5.39	8.51	1.25	0.50	0.61	2.08	0.19
	T4	detik	0.016	0.031	0.016	0	0	0	0.00	0.00
		%	17.21	10.44	8.51	0.00	0.00	0.00	0.00	0.00
	T5	detik	0	0	0	0.02	0.03	40.16	0.00	0.48
		%	0	0.00	0.00	0.01	0.01	10.30	0.00	0.03
	T6	detik	0.031	0.063	0.047	20.98	69.25	21.09	6.63	790.39
		%	33.33	21.21	25.00	14.93	17.26	5.41	25.90	41.73
	Tot	detik	0.094	0.297	0.188	140.53	401.23	389.84	25.58	1894

Pada Tabel 5.11 tersebut terlihat bahwa algoritma penelitian ini memerlukan waktu paling banyak untuk proses pencarian titik inti, bahkan untuk

set data besar proses ini menghabiskan lebih dari 80% total waktu. Hal ini sesuai dengan kompleksitas waktu proses ini, yang telah dibahas pada sub-bab 3.2 yaitu $O(N*((N-1)*(d+(MinPts-1)^2)-(MinPts-1)^3))+N_I)$.

Tabel 5.12 menampilkan kompleksitas waktu algoritma proses pencarian titik inti pada masing-masing set data. Pada tabel itu jelas terlihat bahwa dalam proses ini jumlah data sangat berpengaruh pada kompleksitas waktunya.

Tabel 5.12. Pengaruh Jumlah Data pada Kompleksitas Waktu Proses Pencarian Titik Inti

Set Data	Jumlah Data (N)	Jumlah Atribut (d)	MinPts	Jumlah Titik Inti (N_I)	Kompleksitas Waktu
Iris	150	4	4	114	$3.04x10^5$
			17	39	$5.20x10^6$
Wine	177	13	4	111	$7.00x10^5$
			17	115	$7.68x10^6$
New Thyroid	215	5	4	147	$6.70x10^5$
			17	161	$1.12x10^7$
SAT	4435	36	4	4432	$9.04x10^8$
			17	4432	$5.74x10^9$
Pendigit	10992	16	4	5625	$3.08x10^9$
			17	7007	$3.29x10^{10}$
Shuttle	14500	9	4	14490	$3.99x10^9$
			17	14490	$5.59x10^{10}$
Image-Seg	2100	19	4	1406	$1.26x10^8$
			17	1509	$1.21x10^9$
Letter	20000	16	4	11176	$1.02x10^{10}$
			17	12269	$1.09x10^{11}$

Selain itu terlihat bahwa proses k-Means pada titik representasi menyebabkan iterasi pada klasterisasi akhir lebih sedikit. Semakin banyak jumlah iterasi k-Means titik representasi maka iterasi pada k-Means akhir akan semakin berkurang. Sesuai dengan pembahasan pada bab 3, kompleksitas waktu proses penentuan titik pusat awal klaster adalah $O(N_g^2+(k*d)*(1+i_{TR}*N_g)+N_I)$ dengan kompleksitas waktu untuk kasus terbaik (*best-case*) $O(k*d + N_I)$ terjadi bila jumlah grup sama dengan jumlah klaster. Sedangkan kompleksitas waktu proses penentuan akhir titik pusat klaster (klasterisasi titik data) adalah $O(i_a*k*N*d)$ dengan kompleksitas waktu kasus terbaik 0 (nol), yaitu bila semua titik adalah titik inti.

Besarnya kompleksitas waktu penentuan titik pusat awal klaster dipengaruhi oleh banyaknya jumlah titik representasi grup (N_G) Jika jumlah titik

representasi sedikit, akan sangat menguntungkan apabila proses ini beriterasi lebih sering daripada iterasi pada klasterisasi akhir. Namun jika jumlah titik representasi banyak, maka setiap iterasi akan memakan waktu yang lebih banyak. Hal ini terlihat pada set data SAT dan Shuttle. Bahkan pada set data SAT, meskipun jumlah iterasi k-Means pada titik representasi lebih sedikit, namun kompleksitas waktunya lebih besar. Tabel 5.13 berikut menampilkan perbandingan kompleksitas waktu klasterisasi k-Means pada titik representasi dan k-Means akhir yang dipengaruhi oleh jumlah titik representasi.

Tabel 5.13. Perbandingan Kompleksitas Waktu Penerapan *k*-Means pada Titik Representasi (T5) dan *k*-Means Akhir (T6)

Set Data	MinPts	N_I	N_G	i_{TR}	i_a	Komp. Waktu T5	Komp. Waktu T6
Iris	4	114	30	3	8	2094	1.44×10^4
	17	39	3	0	12	51	2.16×10^4
Wine	4	111	18	6	5	4647	3.45×10^4
	17	115	3	0	6	154	4.14×10^4
New Thyroid	4	147	8	2	10	451	3.23×10^4
	17	161	3	0	8	176	2.58×10^4
SAT	4	4432	4432	11	12	3.19×10^7	1.34×10^7
	17	4432	3319	2	14	2.18×10^7	1.56×10^7
Pendigit	4	5625	584	10	20	1.28×10^6	3.52×10^7
	17	7007	26	4	29	2.43×10^4	5.10×10^7
Shuttle	4	14490	14497	16	15	2.25×10^8	1.37×10^7
	17	14490	14484	29	10	2.36×10^8	9.14×10^6
Image-Seg	4	1406	171	11	44	2.81×10^5	1.23×10^7
	17	1509	7	0	17	1642	4.75×10^6
Letter	4	11176	2496	82	100	9.14×10^7	8.32×10^8
	17	12269	108	8	75	3.83×10^5	6.24×10^8

Algoritma k-Means berpartisi sederhana melakukan inisiasi titik pusat awal klaster secara sembarang (*random*), akibatnya hasil yang diperoleh setiap kali algoritma ini dijalankan berubah-ubah. Oleh karena itu uji coba kali ini menerapkan algoritma k-Means berpartisi sederhana sebanyak dua puluh kali pada setiap set data, dengan jumlah pembagi (*splitting number)* 11, batas error 20, dan batas maksimum iterasi 100. Tabel 5.14 menunjukkan kesalahan klasterisasi dari dua puluh kali penerapan algoritma beserta rerata dan standar deviasi pada setiap set data. Rerata kesalahan klasterisasi itulah yang akan digunakan dalam uji coba perbandingan ini.

Tabel 5.14. Kesalahan Klasterasi dari 20 kali Penerapan Algoritma k-Means Berpartisi Sederhana untuk Setiap Set Data

	Kesalahan Klasterisasi (dalam %)							
	Iris	**Wine**	**New Thyroid**	**Sat**	**Pendigit**	**Shuttle**	**Image Seg**	**Letter**
1	10.67	29.94	37.21	42.5	32.05	33.48	62	78.42
2	10.67	29.94	13.95	42.89	37.29	19.33	51.14	75.51
3	11.33	29.94	15.81	46.29	23.42	31.94	47.81	78.17
4	12.00	29.94	37.21	29.74	40.97	31.54	53.38	76.15
5	11.33	29.94	15.81	37.27	33.26	15.79	63.48	77.1
6	11.33	33.33	35.35	32.47	30.73	17.66	51.57	78.31
7	22.67	28.81	37.21	41.24	23.24	33.09	48.76	77.95
8	11.33	29.94	37.21	29.52	30.86	31.75	45.1	79.69
9	8.67	31.64	37.21	41.24	33.42	15.95	49.57	79.39
10	11.33	28.81	37.67	29.52	36.95	34.45	63.43	78.94
11	10.67	28.81	13.95	44.94	23.56	41.30	60.24	79.18
12	11.33	29.94	43.72	44.42	38.15	28.04	45.48	77.04
13	10.67	29.94	37.21	44.22	26.23	31.32	49.14	78.39
14	22.00	29.94	13.95	32.47	35.40	33.96	53	77.24
15	11.33	29.94	21.86	42.05	33.96	33.61	53.67	79.13
16	49.33	29.94	37.21	45.48	34.65	31.21	52.57	77.39
17	44.00	32.20	15.81	35.24	39.13	33.49	55.86	78.82
18	44.00	27.68	37.21	44.33	33.32	35.60	45.14	77.71
19	11.33	27.68	13.49	37.2	36.64	17.57	49.9	77.21
20	12.00	31.64	13.49	43.36	34.94	33.77	43.76	78.89
Rerata	17.40	30.00	27.12	39.15	32.95	29.02	52.25	78.03
StDev	12.76	1.38	11.62	5.85	5.24	7.54	6.08	1.11

Tabel 5.15. Waktu Komputasi dari 20 kali Penerapan Algoritma k-Means Berpartisi Sederhana untuk Set Data Kecil

Uji ke-	**Iris**			**Wine**			**New Thyroid**		
	it1	**it2**	**detik**	**it1**	**it2**	**detik**	**it1**	**it2**	**detik**
1	1	100	0.156	100	100	1.344	100	100	0.625
2	1	100	0.156	100	100	1.359	100	100	0.641
3	1	100	0.156	100	100	1.344	100	100	0.641
4	1	100	0.203	100	100	1.344	100	100	0.563
5	1	100	0.156	100	100	1.359	100	100	0.641
6	1	100	0.203	100	100	1.359	100	100	0.609
7	1	1	0	100	100	1.359	100	100	0.563
8	1	100	0.156	100	100	1.344	100	100	0.609
9	1	1	0	100	100	1.328	100	100	0.609
10	1	100	0.172	100	100	1.359	100	100	0.609
11	1	100	0.141	100	100	1.359	100	100	0.656
12	1	100	0.172	100	100	1.328	100	100	0.625

13	1	100	0.156	100	100	1.313	100	100	0.547
14	1	100	0.219	100	100	1.313	100	100	0.656
15	1	100	0.172	100	100	1.344	100	100	0.438
16	1	100	0.172	100	100	1.359	100	100	0.635
17	1	100	0.141	100	100	1.328	100	100	0.641
18	1	100	0.141	100	100	1.328	100	100	0.609
19	1	100	0.156	100	100	1.359	100	100	0.625
20	1	1	0.016	100	100	1.344	100	100	0.641
Rerata			0.142			1.344			0.609
StDev			0.063			0.016			0.05
Total Waktu			2.844			26.87			12.18

Tabel 5.16. Waktu Komputasi dari 20 kali Penerapan Algoritma k-Means Berpartisi Sederhana untuk Set Data Besar

Uji ke-	SAT			Pendigit			Shuttle			Image-Seg			Letter		
	it1	it2	dtk	it1	it2	dtk	it1	it2	dtk	it1	it2	dtk	it1	it2	dtk
1	100	100	94.56	100	100	203	100	100	1207	100	100	25.59	100	100	356
2	100	100	94.3	100	100	208	100	100	2206	100	100	20.03	100	100	405
3	100	100	94.05	100	100	191	100	100	1010	100	100	23.29	100	100	323
4	100	100	94.2	100	100	217	100	100	1013	100	100	21.77	100	100	362
5	100	100	96.91	100	100	209	100	100	1561	100	100	21.78	100	100	378
6	100	100	96.5	100	100	203	100	100	1690	100	100	20.92	100	100	351
7	100	100	94.31	100	100	190	100	100	1191	100	100	20.20	100	100	341
8	100	100	96.39	100	100	204	100	100	1072	100	100	21.45	100	100	376
9	100	100	97.67	100	100	210	100	100	1662	100	100	21.80	100	100	383
10	100	100	97.69	100	100	208	100	100	1167	100	100	24.21	100	100	373
11	100	100	94.45	100	100	193	100	100	830.5	100	100	25.83	100	100	429
12	100	100	94.81	100	100	226	100	100	1242	100	100	36.38	100	100	363
13	100	100	98.83	100	100	187	100	100	1160	100	100	38.95	100	100	358
14	100	100	95.98	100	100	215	100	100	1173	100	100	42.78	100	100	356
15	100	100	97.38	100	100	210	100	100	1184	100	100	27.78	100	100	392
16	100	100	93.92	100	100	207	100	100	1146	100	100	33.25	100	100	371
17	100	100	93.72	100	100	207	100	100	1180	100	100	29.69	100	100	352
18	100	100	94.02	100	100	210	100	100	1162	100	100	21.56	100	100	356
19	100	100	98.97	100	100	209	100	100	1728	100	100	22.14	100	100	347
20	100	100	94.33	100	100	213	100	100	1180	100	100	22.48	100	100	378
Rerata			95.65			205.9			1288			26.09			367
StDev			1.748			9.6			320.6			6.70			23
Total			1913			4119			25763			521.89			7349

Tabel 5.15 dan 5.16 diatas adalah hasil dua puluh kali penerapan algoritma pembanding tersebut pada set data kecil (Table 5.15) dan set data besar (Tabel

5.16) yang menginformasikan banyaknya iterasi penerapan k-Means pada set data sederhana (it1), iterasi penerapan k-Means pada BUB dan BUNB (it2), waktu komputasi, serta rerata, standard deviasi, dan total waktu komputasi.

Perbandingan performa kinerja antara algoritma dalam penelitian ini dengan algoritma k-Means berpartisi sederhana ditunjukkan oleh Tabel 5.17 dan 5.18 di bawah ini :

Tabel 5.17 Perbandingan Kesalahan Klasterisasi (dalam persen) antara kedua algoritma.

	Kesalahan Klasterisasi (%)				
	Algoritma Penelitian		*k*-Means Partisi Sederhana	Perbaikan	
	***MinPts*=4**	***MinPts*=17**		***MinPts*=4**	***MinPts*=17**
Iris	11.33	11.33	17.40	6.07	6.07
Wine	29.94	29.94	30.00	0.06	0.06
New Thyroid	15.81	15.35	27.12	11.31	11.77
Sat	35.6	29.65	39.15	3.55	9.5
Pendigit	26.1	31.07	32.95	6.85	1.88
Shuttle	29.04	29.0	29.02	-0.02	0.02
Image Seg	39.62	46.62	52.25	12.63	19.63
Letter	74.58	74.84	78.03	3.45	3.19
			Rerata :	5.49	6.52

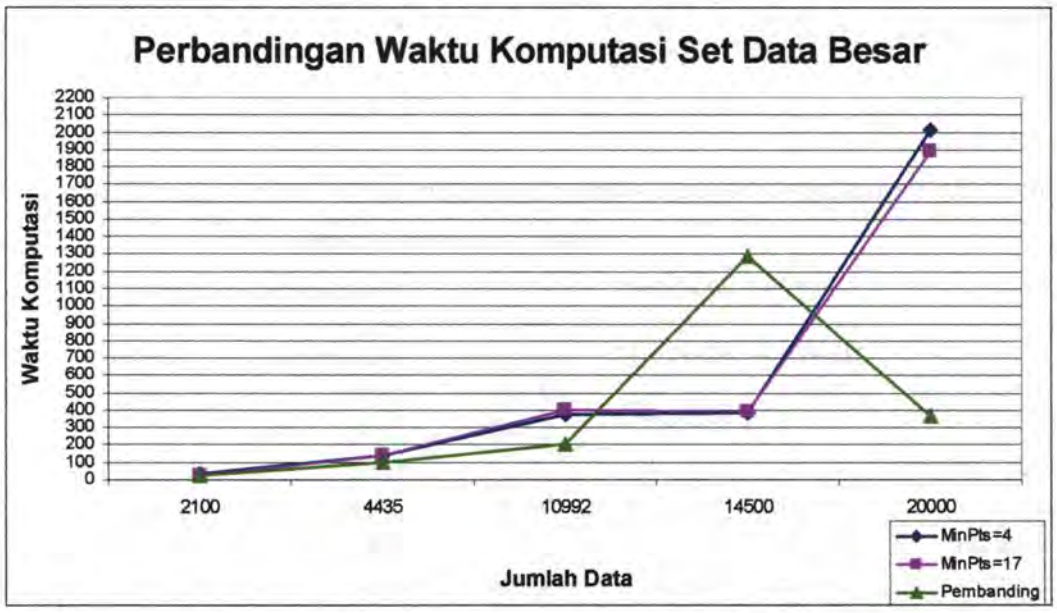
Tabel 5.18 Perbandingan Waktu Komputasi (dalam detik) antara kedua algoritma.

	Waktu Komputasi (detik)						
	Algoritma Penelitian		*k*-Means Partisi Sederhana	Selisih (detik | kali lipat)			
	***MinPts*=4**	***MinPts*=17**		***MinPts*=4**		***MinPts*=17**	
Iris	0.094	0.094	0.142	0.048	1.5	0.048	1.5
Wine	0.203	0.297	1.344	1.141	6.6	1.047	4.5
New Thyroid	0.156	0.188	0.609	0.453	4	0.421	3.2
Sat	135.438	140.531	95.65	-39.788	0.7	-44.881	0.7
Pendigit	374.25	401.234	205.9	-168.35	0.6	-195.33	0.5
Shuttle	379.984	389.844	1288	908.016	3.4	898.156	3.3
Image Seg	34.59	25.58	26.09	-8.50	0.75	0.51	1.02
Letter	2010	1894	367	-1643	0.18	-1527	0.19

Performa kualitas algoritma penelitian ini, baik dengan *MinPts* 4 maupun 17, secara umum lebih baik daripada algoritma k-Means berpartisi sederhana. Meskipun pada set data Shuttle algoritma penelitian dengan *MinPts* 4 salah

mengklaster lebih banyak daripada algoritma pembanding dengan selisih 0.02%, namun pada set-set data lain kualitas algoritma penelitian ini, dengan *MinPts* 4, jauh lebih baik daripada algoritma pembanding. Dengan *MinPts* 17, performa kualitas algoritma penelitian ini selalu lebih unggul daripada algoritma k-Means berpartisi sederhana. Rerata perbaikan kualitas algoritma penelitian ini dengan dua *MinPts* tersebut adalah 6%.

Pada set data kecil, algoritma penelitian ini bekerja lebih cepat, antara 1.5 hingga 6.6 kali lipat, dari algoritma pembanding. Pada set data Image-Seg yang datanya berjumlah 2100, kecepatan algoritma pembanding mulai mengalahkan algoritma penelitian ini. Meski terjadi anomali pada set data Shuttle, yaitu algoritma yang dikembangkan lebih cepat daripada algoritma pembanding, namun pada umumnya seiring dengan pertambahan jumlah data kecepata algoritma penelitian ini akan semakin lambat dan jauh tertinggal dengan algoritma pembanding. Grafik pada Gambar5.15 memperlihatkan analisa ini.



Gambar 5.15 Perbandingan Waktu Komputasi pada Set Data Besar

BAB 6
PENUTUP

6.1 Simpulan

Dari hasil uji coba dan evaluasinya, maka dapat ditarik beberapa simpulan seperti berikut :

1. Algoritma yang dikembangkan dalam penelitian ini membutuhkan nilai *MinPts* yang berasal dari luar sistem, yaitu berupa masukan dari pengguna. Dalam penelitian ini telah dilakukan uji coba untuk mencari nilai *MinPts* yang optimal untuk kedelapan set data yang tersedia dan menghasilkan nilai 4 dan 17 sebagai nilai *MinPts* optimal. Kedua nilai *MinPts* itu kemudian digunakan sebagai masukan algoritma untuk melakukan uji coba perbandingan dengan algoritma *k*-Means berpartisi sederhana yang dilakukan Hung, dkk (Hung, 2005).
2. Algoritma yang dikembangkan dalam penelitian ini menghasilkan akurasi yang lebih baik daripada hasil algoritma *k*-means berpartisi sederhana. Dalam uji coba perbandingan yang dilakukan, algoritma ini memiliki rerata perbaikan dibandingkan kualitas algoritma *k*-Means berpasrtisi sederhana sebesar 5,49% untuk algoritma dengan *MinPts* 4, dan 6,52% untuk algoritma dengan *MinPts* 17.
3. Waktu komputasi yang diperlukan oleh algoritma dalam penelitian ini meningkat seiring dengan bertambahnya jumlah data. Untuk set data kecil, algoritma ini bekerja lebih cepat dibandingkan algoritma *k*-Means berpartisi sederhana (dengan kondisi batas kesalahan 20 dan maksimum iterasi 100). Sedangkan untuk set data besar secara umum algoritma yang dikembangkan ini lebih lambat dibandingkan algoritma *k*-Means berpartisi sederhana.
4. Walaupun perbaikan akurasi yang dihasilkan relatif kecil (5,49% untuk penentuan dengan *MinPts*=4 dan 6,52 untuk *MinPts*=17), algoritma dalam penelitian ini mampu memberikan hasil klasterisasi yang pasti sehingga algoritma ini hanya perlu dijalankan sekali saja. Hal ini berbeda dengan algoritma *k*-means berpartisi sederhana yang menggunakan metoda random

dalam inisiasi awal titik pusat klasternya sehingga harus dijalankan berulang kali agar memperoleh hasil yang pasti (tidak untung-untungan).

6.2 Saran

Lambatnya waktu komputasi yang diperlukan algoritma yang dikembangkan dalam penelitian ini seiring dengan pertambahan jumlah data dikarenakan algoritma membutuhkan kompleksitas waktu yang cukup tinggi pada saat melakukan pencarian titik inti. Pencarian titik inti tersebut menggunakan metode ketetanggaan terdekat yang mengharuskan algoritma menghitung jarak antar titik di tiap data. Akibatnya pada saat data yang digunakan semakin besar, maka waktu yang dibutuhkan untuk penghitungan ketetanggaan terdekat pun semakin lama. Dalam uji coba terbukti bahwa untuk set data besar, waktu yang digunakan algoritma untuk mencari titik inti dapat mencapai 90% dari keseluruhan waktu komputasi. Oleh karena itu, pengembangan metode untuk mengurangi kompleksitas waktu dari algoritma ini masih sangat terbuka.

DAFTAR PUSTAKA

Alsabti, K., Ranka, S., dan Singh, V. (1998), "An Efficient K-Means Clustering Algorithm", *Proceedings of 1st Workshop on High Performance Data Mining.*

Breunig, M.M., Kriegel, H.P., Ng, R.T., dan Sander, J. (2000), "LOF: Identifying Density Based Local Outliers", *Proceeding ACM 2000 International Conference on Management of Data,* Dalles, Texas, SIGMOD'00, hal.93-104

Ester, M., Kriegel, H.P, Sander, J., dan Xu, X. (1996), "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise", *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining,* Portland, Oregon, hal. 226-231

Fahim, A.M., Salem, A.M., Torkey, F.A., dan Ramadan M.A. (2006), "An Efficient Enhanced K-means Clustering Algorithm, *Journal of Zheijian University SCIENE A,* ISSN 1862-1775 (online).

Feng, Y. dan Hamerly, G. (2006), *Learning the Number of Clusters in a Dataset*, Technical Report, Computer Sciene Departement, Baylor University, Waco, Texas, American Association for Artificial Intelligence.

Hung, M.C., Wu, J., Chang, J.H., dan Yang, D.I. (2005), "An Efficient K-means Clustering Algorithm Using Simple Partitioning", *Journal of Information Sciene and Engineering 21,* hal. 1157-1177.

Kanugo, T., Netanyahu, N.S., Piatko, C.D., Silverman, R., dan Wu, A.Y. (2002), "An Efficient K-means Clustering Algorithm: Analysis and Inplementation", *IEEE Transaction on Pattern Analysis and Machine Intelligence*,24(7), hal 881-892.

Martiana, E. dan Djunaidy, A. (2006), *Penerapan Algoritma Genetika Cepat dalam Perbaikan Penentuan Titik Pusat Awal pada Klasterisasi Berbasis K-means*, Tesis Master, Institut Teknologi Sepuluh November, Surabaya, Indonesia.

Mualik, U., Bandyopadhyay, S., dan Pakhira, M.K. (2001), "Clustering Using Simulated Annealing with Probabilistic Redistribution", *Int. J. Pattern Recognition and Artificial Intelligence*, vol. 15(2), hal. 269-285.

Pham, D.T., Dimov, S.S., dan Nguyen, C.D. (2004), "An Icremental K-Means Algorithm", *Proceedings Instn. Mech. Engrs*, Journal Mechanical Engineering Science, Vol. 218 bag. C, hal. 783-795

Tan, P.N., Steinbach, M., dan Kumar, V. (2006), "*Introduction to Datamining*", Pearson-Addison Wesley, Boston, USA