Clustering Tagg Status Facebook Dengan Menggunakan Algoritma K-MEDOIDS

Sefia Candra[1]
sefia_chandra@yahoo.com

Antonius R.C[2]
anton@ti.ukdw.ac.id

Lucia Dwi Krisnawati[3]
krisna@ukdw.ac.id

Abstract

This research is implementing K-Medoids algorithm to discover clusters on a friend list of a Facebook user. To find those clusters, the system uses the strongest path which is based on the tag frequency of status update of the facebook user to measure the tie strength from a friend to other friends. The experiments of using 3 clusters, 5 clusters, and 7 clusters, which resulted in average purity score 0.7430. The experiment resulted in rank of highest average purity score, at the first rank is experiment which used 3 clusters with the average score 0.8806, at the second rank is experiment which used 7 clusters with the average score 0.7114, and the third rank is experiment which used 5 clusters with the average score 0.6368.

Keywords: *cluster, Dijkstra, Facebook, strongest path, K-Medoids, purity, status update, tag*

1. PENDAHULUAN

Facebook merupakan jejaring sosial yang terkenal di dunia yang membantu pengguna untuk menjalin pertemanan yang sangat luas. Pengguna Facebook dapat menjalin pertemanan dengan ratusan bahkan ribuan teman, baik yang dikenal maupun yang tidak. Akan tetapi, ketika pertemanan menjadi begitu besar, akan sangat sulit untuk memilah-milah informasi, informasi mana yang ingin diterima dan yang akan dibagikan ke teman lain. Facebook sendiri berusaha untuk menyelesaikan masalah tersebut dengan meningkatkan fitur dalam daftar pertemanan, yaitu dengan membuat daftar teman dan mengelompokkan teman secara otomatis dan *up-to-date* berdasarkan informasi pengguna Facebook, seperti berdasarkan sekolah, tempat kerja, keluarga, dan domisili. Namun, pengelompokkan tersebut masih terlalu luas dan tidak dapat menggambarkan suatu kelompok individu yang saling berhubungan baik.

[1] *Teknik Informatika, Fakultas Teknologi Informasi, Universitas Kristen Duta Wacana, Yogyakarta*
[2] *Teknik Informatika, Fakultas Teknologi Informasi, Universitas Kristen Duta Wacana, Yogyakarta*
[3] *Teknik Informatika, Fakultas Teknologi Informasi, Universitas Kristen Duta Wacana, Yogyakarta*

Berdasarkan masalah di atas, pada penelitian ini, penulis akan membangun sebuah aplikasi berbasis algoritma K-medoids untuk menemukan *cluster-cluster* pada daftar teman pengguna Facebook, dengan memanfaatkan frekuensi interaksi *tagging* pada *status update* sebagai dasar nilai kedekatan antara satu teman ke teman yang lain.

2. LANDASAN TEORI

2.1. Algoritma Dijkstra

Algoritma Dijkstra adalah suatu algoritma untuk menemukan jalur terpendek dengan biaya termurah atau jarak terkecil dari sebuah *node* ke semua *node* yang ada pada suatu graf berbobot. Jarak dari antar *node* dilambangkan dengan bobot pada *edge* yang menghubungkan *node* bersangkutan. Jarak antar *node* harus bernilai positif (>= 0).

Langkah-langkah algoritma Dijkstra adalah sebagai berikut (The Department of Land Surveying and Geo-Informatics [LSGI], 2011):

1. Set semua nilai jarak. Set 0 untuk *node* awal dan ∞ untuk semua *node* lain.
2. Tandai semua *node* sebagai *node* yang belum dikunjungi dan set *node* awal sebagai *node* yang dipilih.
3. Untuk *node* yang dipilih, tinjau semua *node* lain yang belum dikunjungi dan hitung jaraknya (dari *node* awal). Jika jarak yang dihasilkan lebih kecil daripada jarak sebelumnya ganti nilai jarak menjadi hasil perhitungan jarak yang baru.
4. Ketika meninjau semua *node* lain pada *node* yang dipilih dan *node* lain sudah pernah dikunjungi, maka tidak perlu dicek kembali nilai jaraknya karena jarak yang tercatat pada *node* yang sudah pernah dikunjungi adalah jarak final dan minimal.
5. Set *node* yang belum pernah dikunjungi yang memiliki jarak terkecil (dari *node* awal) sebagai *node* yang dipilih dan ulangi langkah 3.

2.2. Algoritma Jalur Terkuat

Hangal et al. (2010) mengusulkan sebuah algoritma untuk menemukan dan menghitung nilai jalur terkuat (*strongest path*) dari *node* sumber ke *node* target pada *social network*. Jalur terkuat adalah jalur yang memberi nilai pengaruh (*influence*) tertinggi dari *node* sumber ke *node* target. Pengaruh dimodelkan dengan interaksi yang memerlukan biaya atau investasi, seperti waktu dan usaha yang diperlukan. Besarnya investasi yang diberikan (jumlah interaksi) merupakan ukuran informatif dari *tie strength*. Secara singkat untuk menghitung besarnya *influence* digunakan rumus berikut.

$$\text{Influence}(A, B) = \frac{\text{Invests}(B, A)}{\sum_X \text{Invests}(B, X)} \quad [1]$$

Keterangan:

Influence (A,B)	:	besarnya pengaruh yang diberikan oleh *node A* (sumber) kepada *node B* (target).
Invests (B,A)	:	total interaksi yang dilakukan dari *node B* ke *node A*.
$\sum_x Invests\ (B, X)$	:	total interaksi yang dilakukan dari *node B* ke semua *node* yang ada pada graf.

Untuk menemukan jalur terkuat antar *node*, Hangal et al. (2010) memodelkannya dengan menggunakan graf berbobot dan berarah, di mana *node* mewakili aktor (orang), bobot *edge* mewakili pengaruh, dan arah *edge* mewakili arah pengaruh. Hangal et al. juga menambahkan sebuah nilai probabilitas konstan berupa *discount factor* untuk setiap *edge* karena panjang jalur yang dilalui akan mempengaruhi nilai pengaruh yang diberikan oleh *node* sumber ke *node* target. Sebagai contoh sebuah jalur P dengan panjang |P| yang terdiri dari *edge* e_1, e_2, ..., e_n dan sebuah *discount factor* x, maka kekuatan dari jalur P adalah:

$$\mathrm{S}(P) = \prod D \times \mathrm{Influence}(e_i), e_i \in P. \quad [2]$$

Algoritma ini mengadaptasi algoritma Dijkstra untuk menghitung nilai jalur terkuat antara dua *node*, *node* sumber ke *node* target. Langkah-langkah adaptasi yang dilakukan pada algoritma Dijkstra untuk menghitung jalur terkuat adalah:

- Jalur *P* adalah jalur dari *node* sumber ke *node* target yang dapat memaksimalkan nilai dari rumus [2.2].
- Pada kasus ini, akan dimaksimalkan logaritma pada matriks berikut:

$$\sum log(D) + log(\mathrm{Influence}(e_i)), e_i \in P \quad [3]$$

- Oleh karena itu, untuk meminimaliskannya menggunakan rumus:

$$-\sum log(D) + log(\mathrm{Influence}(e_i)), e_i \in P \quad [4]$$

- Akhirnya, meminimaliskan dengan menggunakan rumus berikut ini:

$$\sum log(1/D) + log(1/\mathrm{Influence}(e_i)), e_i \in P \quad [5]$$

Oleh karena itu, dengan menggunakan algoritma Dijkstra dapat dihitung nilai jalur terkuat, di mana bobot *edge*/jarak pada algoritma Dijkstra adalah hasil perhitungan dari

$$\log(\frac{1}{D}) + \log(\frac{1}{influence(e_i)}) \quad [6]$$

Dengan kata lain, semakin rendah total bobot jalur yang dihasilkan oleh algoritma Dijkstra, maka semakin tinggi nilai kekuatan jalurnya.

2.3. Algoritma K-medoids

Algoritma K-medoids atau dikenal pula dengan *PAM* (*Partitioning Around Medoids*) menggunakan metode partisi *clustering* untuk mengelompokkan sekumpulan *n* objek menjadi sejumlah *k cluster.* Algoritma ini menggunakan objek pada kumpulan objek untuk mewakili sebuah *cluster*. Objek yang terpilih untuk mewakili sebuah *cluster* disebut *medoid*. *Cluster* dibangun dengan menghitung kedekatan yang dimiliki antara *medoid* dengan objek *non-medoid*.

Menurut Han dan Kamber (2006, hal 406) algoritma K-medoids adalah sebagai berikut.

1. Secara acak pilih *k* objek pada sekumpulan *n* objek sebagai *medoid*.
2. Ulangi:
3. Tempatkan objek *non-medoid* ke dalam *cluster* yang paling dekat dengan *medoid*.
4. Secara acak pilih o_{acak}: sebuah objek *non-medoid*.
5. Hitung total biaya, *S*, dari pertukaran medoid o_j dengan o_{random}.
6. Jika $S < 0$ maka tukar o_j dengan o_{acak} untuk membentuk sekumpulan *k* objek baru sebagai *medoid*.
7. Hingga tidak ada perubahan.

2.4. Purity

Menurut Manning, Raghavan, dan Schütze (2009), *purity* merupakan salah satu ukuran untuk mengukur kualitas *clustering* berbasis *external criterion*. *External criterion* adalah metode untuk mengevaluasi seberapa baik hasil *clustering* dengan menggunakan sekumpulan kelas acuan sebagai wakil penilaian pengguna, di mana kelas acuan ini diperoleh dari hasil penilaian manusia. Evaluasi ini tidak menggunakan label kelas pada kelas acuan, tetapi hanya menggunakan hasil partisi pada kelas acuan.

Secara *form*al, untuk menghitung nilai *purity* dapat digunakan rumus berikut.

$$\text{purity}(\Omega, \mathbb{C}) = \frac{1}{N} \sum_{k} \max_{j} |\omega_k \cap c_j| \qquad [7]$$

Keterangan:

$\Omega = \{w_1, w_2, \ldots, w_k\}$ adalah kumpulan cluster

$C = \{c_1, c_2, \ldots, c_j\}$ adalah kumpulan kelas acuan

w_k = adalah kumpulan objek pada *cluster* w_k

c_j = adalah kumpulan objek pada kumpulan kelas acuan c_j

Jika nilai *purity* semakin mendekati 1 menunjukkan bahwa *clustering* memberikan hasil yang semakin baik. Sedangkan, jika nilai *purity* semakin mendekati 0, maka *clustering* memberikan hasil yang semakin buruk.

3. HASIL DAN PEMBAHASAN

3.1. Contoh Perhitungan Manual Sistem

3.1.1. Penyaringan Teman

Dimisalkan, user X telah melakukan proses *login* dan mendapatkan *access token* berupa xxxx. Kemudian, dari daftar teman Facebook yang dimiliki X, X memilih A, B, C, D, E, F, dan G sebagai teman-teman yang ingin dikelompokkan. Selanjutnya, sistem akan mengirimkan *request* ke *Facebook APIs* melalui *FQL* untuk masing-masing teman. Misalnya untuk teman A, maka *request* yang dikirimkan adalah https://graph.facebook.com/fql?q=SELECT target_id FROM stream_tag WHERE actor_id=A&access_token=xxxx.

Berdasarkan hasil pengembalian yang diterima oleh sistem, maka didapatkan frekuensi interkasi *tagging* pada *status update* Facebook yang dilakukan antar teman-teman yang telah dipilih oleh X. Data tersebut dapat dilihat pada bagian (a) Gambar 1 (untuk mempermudah penjelasan data ini akan disebut data pada tabel tagging).

Berdasarkan data pada tabel tagging, teman H pernah dikenai *tag* oleh G, namun H tidak pernah melakukan *tag* ke teman-teman yang lain, maka data pada tabel tagging yang berisikan H dihapus. Sehingga tabel tagging hanya berisikan data-data seperti pada bagian (b) Gambar 1.

(a)

actor	target	tag
A	D	1
B	E	1
C	A	2
D	A	3
D	C	2
E	B	4
E	F	4
F	E	3
G	H	2

(b)

actor	target	tag
A	D	1
B	E	1
C	A	2
D	A	3
D	C	2
E	B	4
E	F	4
F	E	3

(c)

Id_ friend	derajat tag
A	2
B	1
C	1
D	1
E	2
F	1

Gambar 1. Contoh Penyaringan Teman

3.1.2. Perhitungan Bobot *Edge*

Berdasarkan data pada tabel tagging yang telah melalui proses penyaringan, maka didapatkan *influence* dari satu teman ke teman yang lain dengan menggunakan rumus [1] seperti pada bagian (a) Gambar 2. Dari hasil perhitungan *influence* tersebut, dihitung bobot *edge* dari satu teman ke teman yang lain dengan menggunakan rumus [6] dan *discount factor* sebesar 0,95 (sama dengan nilai *discount factor* yang digunakan oleh Hangal et al., 2010). Hasil perhitungan bobot *edge* dapat dilihat pada bagian (b) Gambar 2 (untuk mempermudah penjelasan data ini akan disebut data pada tabel influence).

(a)

souce	target	influence	Keterangan
A	C	1	$\frac{2}{2}$
A	D	0,6	$\frac{3}{3+2}$
B	E	0,5	$\frac{4}{4+4}$
C	D	0,4	$\frac{2}{3+2}$
D	A	1	$\frac{1}{1}$
E	B	1	$\frac{1}{1}$
E	F	1	$\frac{3}{3}$
F	E	0.5	$\frac{4}{\cdots}$

(b)

souce	target	weight	Keterangan
A	C	0,0223	$\log(\frac{1}{0,95}) + \log(\frac{1}{1})$
A	D	0,2441	$\log(\frac{1}{0,95}) + \log(\frac{1}{0,6})$
B	E	0,3233	$\log(\frac{1}{0,95}) + \log(\frac{1}{0,5})$
C	D	0,4202	$\log(\frac{1}{0,95}) + \log(\frac{1}{0,4})$
D	A	0,0223	$\log(\frac{1}{0,95}) + \log(\frac{1}{1})$
E	B	0,0223	$\log(\frac{1}{0,95}) + \log(\frac{1}{1})$
E	F	0,0223	$\log(\frac{1}{0,95}) + \log(\frac{1}{1})$
F	E	0.3233	$\log(\frac{1}{\cdots}) + \log(\frac{1}{\cdots})$

Gambar 2. Contoh Perhitungan Bobot *Edge*

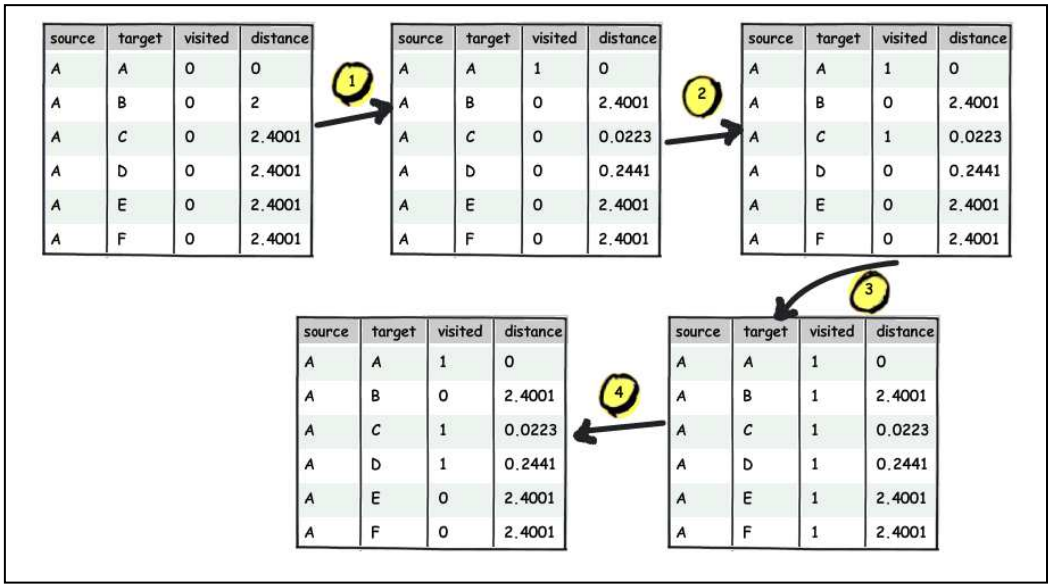
3.1.3. Perhitungan Nilai Jalur Terkuat

Langkah-langkah untuk menghitung jalur terkuat dari satu teman ke teman lain adalah sebagai berikut (untuk mempermudah penjelasan data ini akan disimpan pada tabel strongest):

1. number_weight = (jumlah total pada kolom weight table influence) +1
   = (0,0223 +0,2441 +0,3233 +0,4202 +0,0223 +0,0223 +0,0223 +0,3233) +1
   = 2,4001
2. ls = Data unik pada kolom target tabel influence → {A, B, C, D, E, F}
3. Lakukan perulangan selama ada data pada ls:
   Lakukan perulangan selama ada data pada ls:
   INSERT tabel strongest selama ada data pada ls dengan nilai:
   - (ls, ls, 0, number_weight), jika ls ≠ ls
   - (ls, ls, 0, 0), jika ls = ls
4. Lakukan perulangan selama ada data pada ls:

- Implementasi algoritma Dijkstra

Ilustrasi implementasi algoritma Dijkstra dengan ls= A ditunjukkan pada Gambar 3.



Gambar 3. Ilustrasi Implementasi Algoritma Dijkstra

1. a) Target A dipilih sebagai node terpilih, karena data pada kolom distance merupakan distance terkecil dengan kondisi source=A dan visited=0.
   b) Nilai pada kolom visited tabel strongest dengan kondisi source=A dan target=A diperbarui menjadi 1.
   c) Nilai distance pada target C diganti dengan nilai distance pada target A (0) ditambah dengan nilai pada kolom weight tabel influence dengan kondisi source=A dan target=C (0,0223), karena nilai tersebut lebih kecil dari pada nilai pada kolom distance tabel strongest dengan kondisi source=A dan target =C.
   d) Nilai distance pada target D diganti dengan nilai distance pada target A (0) ditambah dengan nilai pada kolom weight tabel influence dengan kondisi source=A dan target=D (0,2441), karena nilai tersebut lebih kecil dari pada nilai pada kolom distance tabel strongest dengan kondisi source=A dan target =D.
2. a) Target C dipilih sebagai node terpilih, karena data pada kolom distance merupakan distance terkecil dengan kondisi source=A dan visited=0.
   b) Nilai pada kolom visited tabel strongest dengan kondisi source=A dan target=C diperbarui menjadi 1.
3. a) Target D dipilih sebagai node terpilih, karena data pada kolom distance merupakan distance terkecil dengan kondisi source=A dan visited=0.
   b) Nilai pada kolom visited tabel strongest dengan kondisi source=A dan target=D diperbarui menjadi 1.

4. a) Semua nilai kolom visited diperbarui menjadi 1, karena nilai terkecil pada kolom distance dengan kondisi source=A dan visited=0 sama dengan nilai number_weight (jumlah total pada kolom weight table strongest +1).

Hasil akhir nilai jalur terkuat dari satu teman ke teman yang lain merupakan hasil akhir yang tersimpan pada tabel strongest. Hasil akhir pada tabel strongest dapat dilihat pada Gambar 4.

souce	target	visited	distance
A	A	1	0
A	B	1	2,4001
A	C	1	0,0223
A	D	1	0,2441
A	E	1	2,4001
A	F	1	2,4001
B	A	1	2,4001
B	B	1	0
B	C	1	2,4001
B	D	1	2,4001
B	E	1	0,3233
B	F	1	0,3456
C	A	1	0,4425
C	B	1	2,4001
C	C	1	0
C	D	1	0,4202
C	E	1	2,4001
C	F	1	2,4001

souce	target	visited	distance
D	A	1	0,0223
D	B	1	2,4001
D	C	1	0,0446
D	D	1	0
D	E	1	2,4001
D	F	1	2,4001
E	A	1	2,4001
E	B	1	0,0223
E	C	1	2,4001
E	D	1	2,4001
E	E	1	0
E	F	1	0,0223
F	A	1	2,4001
F	B	1	0,3456
F	C	1	2,4001
F	D	1	2,4001
F	E	1	0,3233
F	F	1	0

Gambar 4. Hasil Akhir Tabel Strongest

3.1.4. Pengelompokkan Teman Menggunakan Algoritma K-Medoids

Misalnya daftar teman yang telah melalui proses penyaringan di atas akan dikelompokkan menjadi dua *cluster*, maka langkah-langkah yang akan dilakukan sistem sebagai berikut:

1. Diambil dua teman dengan derajat *tag* tertinggi (dapat dilihat pada bagian (c) Gambar 3.1), sehingga dipilih A sebagai *medoid cluster* pertama (C_1) dengan derajat *tag* 2 yang berasal dari C dan D serta E sebagai *medoid cluster* kedua (C_2) dengan derajat *tag* 2 yang berasal dari B dan F.

   Berdasarkan nilai pada kolom distance tabel strongest, maka didapatkan hasil *cluster*:

   C_1 = {A, C, D} dan C_2 = {E, B, F}

   Total jarak adalah (S_{lama}) = {(A,A) + (A,C) + (A,D) + (E,E) + (E,B) + (E,F)}

   = 0 + 0,0223 + 0,2441 + 0 + 0,0223 + 0,0223 = 0,2910

2. Secara acak, terpilih B sebagai O_{random}, maka A sebagai *medoid cluster* pertama (C_1) dan B sebagai *medoid cluster* kedua (C_2).

   Maka didapatkan hasil *cluster*: C_1 = {A, C, D} dan C_2 = {B, E, F}

   Total jarak adalah (S_{baru}) = {(A,A) + (A,C) + (A,D) + (B,B) + (B,E) + (B,F)}

   = 0 + 0,0223 + 0,2441 + 0 + 0,0223 + 0,3456 = 0,6343

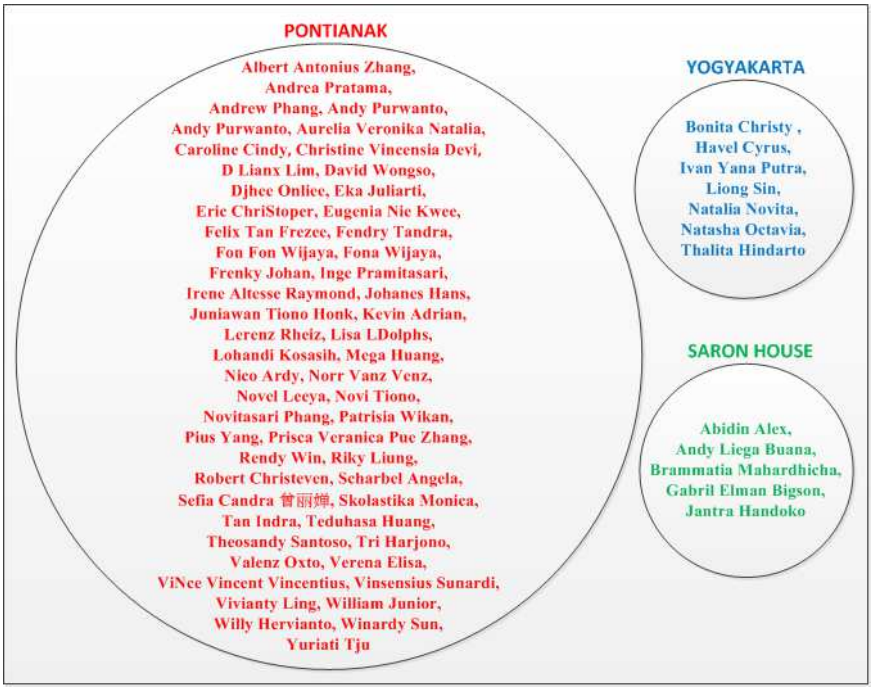
Karena $S_{baru} > S_{lama}$, maka tidak ada pertukaran *medoid*.

3. Lakukan kembali langkah 2 hingga semua teman *non-medoid* terpilih menjadi O_{random} dan tidak terjadi perubahan pada *medoid*.

3.2 Pengujian Sistem

Data yang digunakan dalam pengujian sistem adalah *mutual friends* yang dimiliki oleh tiga pemilik akun Facebook, yaitu Anthony Andrian (ID Facebook 1360992359), Antonius Tornado (ID Facebook 1255512315), dan Gustin Lawis (ID Facebook 1616770571). Berdasarkan tiga akun tersebut didapatkan 175 *mutual friends*. Kemudian dilakukan pengiriman *request* dengan menggunakan *FQL* (*Facebook Query Language*) ke *Facebook APIs* untuk mendapatkan data frekuensi *tag* yang pernah dilakukan antar *mutual friends* (pengambilan data *mutual friends* dan frekuensi *tag* dilakukan tanggal 26 April 2012 20:50). Berdasarkan data frekuensi *tag* yang didapatkan, tersaring 67 *mutual friends*, yaitu teman-teman yang pernah melakukan *tag* ke *mutual friends* yang lain.

Kemudian penulis melakukan diskusi dengan ketiga pemilik akun untuk melakukan pengelompokkan secara manual terhadap 67 *mutual friends* tersebut. Pengelompokkan dilakukan dengan mengelompokkan 67 *mutual friends* tersebut menjadi 3, 5, dan 7 kelompok. Hasil pengelompokkan ini akan digunakan sebagai kelas acuan untuk menghitung nilai *purity* sistem, di mana untuk setiap jumlah kelompok akan dilakukan tiga kali percobaan dikarenakan terdapat unsur *random* pada iterasi algoritma K-Medoids. Hasil pengelompokkan disajikan pada Gambar 5, Gambar 6, dan Gambar 7.



Gambar 5. Kelas Acuan dengan Tiga Kelompok

Gambar 6. Kelas Acuan dengan Lima Kelompok

Gambar 7. Kelas Acuan dengan Tujuh Kelompok

Berdasarkan pengujian yang telah dilakukan, didapatkan rata-rata *purity* untuk masing-masing jumlah *cluster* seperti pada Tabel 1.

Tabel 1.
Rata-Rata *Purity* Hasil Pengujian

Jumlah *Cluster*	Pengujian Ke-	*Purity*	Rata-Rata *Purity*
1	1	0,8806	0,8806
	2	0,8806	
	3	0,8806	
2	1	0,6269	0,6368
	2	0,6269	
	3	0,6567	
3	1	0,7164	0,7114
	2	0,6866	
	3	0,7313	
		Rata-Rata	**0,7430**

4. Kesimpulan

Berdasarkan analisis dan implementasi sistem, maka diperoleh kesimpulan sebagai berikut:

a) Dengan menggunakan algoritma K-Medoids dan nilai jalur terkuat yang didapatkan dari frekuensi *tag* pada *status update* sebagai nilai kedekatan antar teman, sistem telah mampu menemukan *cluster-cluster* dalam daftar teman Facebook dengan baik.
b) Dengan uji coba menggunakan jumlah *cluster* sebanyak 3, 5, dan 7 didapatkan rata-rata nilai *purity* 0,7430.
c) Peringkat rata-rata nilai *purity* tertinggi, yaitu tiga jumlah *cluster* dengan rata-rata 0,8806, tujuh jumlah *cluster* dengan rata-rata 0,7114, dan lima jumlah *cluster* dengan rata-rata 0,6368.

Daftar Pustaka

CodePlex. *JSON.Net Serialize All The Things*. (2012). Diakses 19 Maret 2012, dari http://json.codeplex.com.

Hangal, S., MacLean, D., Lam, M. S., & Heer J. (2010). *All Friends are Not Equal: Using Weights in Social Graphs to Improve Search*. Diakses 19 Maret 2012, dari http://xenon.stanford.edu/~hangal/weighted-social-graphs.pdf.

Han, J., & Kamber, M. (2006). *Data Mining Concepts and Techniques Second Edition*. San Fransico: Morgan Kaufmann Publishers.

Lange, R., Dvornik, T., Hamilton, W., & Hess B. (n.d.). *Data Mining for Facebook Cliques*. Diakses 26 Februari 2012, dari http://www.amphro.com/560_Mine_Cliques_Final.pdf.

Lanzi, P., L., (2009). *Clustering: Partitioning Methods*. Diakses 01 Mei 2012, dari http://www.pierlucalanzi.net/wp-content/teaching/dmtm/DMTM0809-07-ClusteringPartitioning.pdf.

LSGI. (2011). *Shortest Path Algorithm*. Diakses 08 April 2012, dari http://www.lsgi.polyu.edu.hk/staff/Bo.Wu/teaching/lsgi521/11-12/lsgi521_lecture_slides/LSGI521_L6_Spatial%20Analysis%20in%20GIS_Part2_BW.pdf.

Manning, C. D., Raghavan, P., Schütze, H. (2009). *An Introduction to Information Retrieval*. England: Cambridge University Press.

Safitri, A., & Hardani, W. (Ed.). (2005). *Aljabar Linear Elementer Versi Aplikasi*. Diakses 4 April 2012, dari http://books.google.co.id/books?id=v57mQQcr1L8C&pg=PA193&lpg=PA193&dq=clique+adalah&source=bl&ots=TX6Ief2fEA&sig=80Sn7QuVRBod8Ze06a-ofLRn8&hl=id&sa=X&ei=1p6T4uhCYKIrAfuwYinAg&ved=0CGEQ6AEwCQ#v=onepage&q&f=false.

TechTarget. (2008). *Facebook Status*. Diakses 3 April 2012, dari http://whatis.techtarget.com/definition/facebook-status.html.

Wibisono, Y. (2011). *Perbandingan Partition Around Medoids (PAM) dan K-Means Clustering Untuk Tweets*. Diakses 23 Maret 2012, dari http://cs.upi.edu/uploads/yudiwbs/Yudi_Wibisono_UPI_Perbandingan%20Clustering%20PAM%20dan%20KMeans%20untuk%20Posting%20Tweet.pdf.