KOMPUTASI BETWEENNESS CENTRALITY DENGAN ALGORITMA EKSAK DAN PENDEKATAN

Ronsen Purba

STMIK Mikroskil
Jl. Thamrin No. 112, 124, 140 Medan 20212
ronsen@mikroskil.ac.id

Abstrak

Betweenness merupakan ukuran *centrality* berdasarkan pada lintasan terpendek yang banyak digunakan dalam analisis terhadap graf yang kompleks. Solusi eksak terbaik untuk menghitung betweenness centrality adalah O(*nm*) untuk graf tak berbobot dan O($nm+n^2 \log n$) untuk graf berbobot untuk jumlah vertex adalah *n* serta jumlah edge adalah *m*. Dalam dunia nyata dimana ukuran graf sangat besar perhitungan eksak akan menjadi sangat tidak fisibel. Untuk kondisi tersebut maka solusi pendekatan menjadi pilihan terbaik yang dapat dilakukan. Dalam paper ini akan dibandingkan solusi eksak dengan solusi pendekatan untuk menghitung *betweennes centrality*. Algoritma eksak yang digunakan adalah algoritma Brandes yang dieksekusi secara *stand alone* dan paralel serta algoritma pendekatan dengan teknik sampling yang adaptif untuk mengurangi secara signifikan perhitungan jumlah SSSP (*single-source shortest path*) untuk verteks-verteks yang mempunyai nilai *centrality* yang tinggi. Dari ketiga algoritma tersebut akan diberikan diskusi terkait dengan kinerja masing-masing dengan graf uji yang berbeda-beda. Untuk keperluan saat ini dan ke depan dibutuhkan algoritma dinamis dengan kompleksitas lebih baik.

Kata Kunci: *betweenness centrality, algoritma eksak, shortest path, teknik sampling adaptif*

1. Pendahuluan

Salah satu permasalahan mendasar dalam analisis jaringan (*network analysis*) adalah menentukan pentingnya (*centrality*) terhadap vertex tertentu dalam jaringan tersebut. Ukuran-ukuran *centrality* adalah *closeness*, *stress*, dan *betweenness* [1, 9]. Dari semua ukuran tersebut, *betweenness centrality* telah menjadi kajian mendalam dan digunakan dalam menganalisis jaringan sosial dan jaringan kompleks lainnya. *Betweennes* digunakan dalam menganalisis penyakit menular seperti penyakit kelamin dan AIDS, keracunan dalam biologi, menentukan peran utama dalam jaringan teroris, perilaku berorganisasi, dan proses dalam manajemen rantai pasokan. *Betweenness* juga digunakan sebagai rutin utama dalam algoritma populer untuk identifikasi dan pengelompokan (*clustering*) dalam jaringan dunia nyata [13]. Sebagai contoh Girvan-Newman [10] mengemukakan algoritma yang secara iteratif melakukan partisi sebuah jaringan dengan mengidentifikasikan edge dengan nilai *betweenness* yang tinggi, menghapus edge tersebut dan kemudian menghitung ulang nilai *centrality*.

Betweenness adalah ukuran centrality yang bersifat global yang didasarkan pada enumerasi lintasan terpendek. Misalkan $G = (V, E)$, adalah graf berarah atau tidak berarah dimana V adalah himpunan verteks yang merepresentasikan aktor, dan E adalah himpunan garis (edge) yang merepresentasikan hubungan antar aktor. Jumlah verteks dan edge dinotasikan dengan n dan m secara berturut-turut. Kita asumsikan bahwa setiap edge $e \in E$ mempunyai bobot $w(e)$ bilangan bulat positif. Untuk graf tak berbobot kita menggunakan $w(e) = 1$. Sebuah lintasan dari verteks s

ke t didefinisikan sebagai barisan edge $<u_i, u_{i+1}>$, $0 \leq i \leq k$ dimana $u_0 = s$ dan $u_k = t$. Panjang sebuah lintasan adalah jumlah bobot dari edge yang membentuk lintasan tersebut. Kita menyatakan $d(s, t)$ adalah jarak antara verteks s dan t (panjang minimum lintasan sembarang yang menghubungkan s dan t dalam G). Misalkan jumlah lintasan terpendek antara verteks s dan t adalah λ_{st}, dan jumlah lintasan yang melalui verteks v adalah $\lambda_{st}(v)$. Misalkan lagi kita notasikan $\delta_{st}(v)$ untuk menyatakan segmen dari lintasan terpendek antara s dan t yang melewati verteks v, yakni, $\delta_{st}(v) = \frac{\lambda st(v)}{\lambda st}$. Kemudian kita menyatakan $\delta_{st}(v)$ adalah *pair-dependency* dari s, t pada verteks v. *Betweenness centrality* dari verteks v didefenisikan dengan persamaan berikut:

$$C_B(v) = \sum_{s \neq v \neq t \in V} \delta_{st}(v)$$

Saat ini, algoritma eksak tercepat untuk menghitung *betweenness centrality* dari semua verteks, dikembangkan oleh Brandes [6], membutuhkan waktu paling tidak $O(nm)$ untuk graf tak berbobot dan $O(nm+n^2 \log n)$ untuk graf berbobot, dimana n dan m menyatakan jumlah verteks dan jumlah edge secara berturut-turut. Jadi, untuk graf skala besar, solusi eksak untuk *betweenness centrality* dengan kemampuan hardware saat ini akan menjadi sangat tidak praktis. Sebelumnya, Bader & Madduri [2] mengembangkan sebuah algoritma paralel untuk meningkatkan waktu komputasi yang diujikan pada sejumlah graf dalam dunia nyata termasuk jaringan skala besar. Dengan algoritma paralel ditunjukkan ukuran graf dapat ditingkat tiga kali lipat dari ukuran jaringan yang dapat diselesaikan oleh paket *social network analysis* (SNA) saat ini. Hal ini dicapai dengan menggunakan strategi pemilihan verteks awal.

Estimasi yang cepat untuk mendekati perhitungan *betweenness centrality* yang eksak menjadi penting. Akan tetapi algoritma eksak untuk menghitung *centrality* berdasarkan lintasan terpendek akan membutuhkan n buah lintasan terpendek. Akan sangat baik jika dapat dilakukan estimasi untuk hanya menghitung jumlah lintasan terpendek yang lebih sedikit dengan menggunakan ekstrapolasi. Dengan menggunakan teknik sampling yang acak, Eppstein & Wang [8] menunjukkan bahwa *closeness centrality* untuk semua verteks dalam graf berbobot dan tak berarah dapat didekati dengan probabilitas tinggi dalam waktu $O(\frac{\log n}{\epsilon \hat{} 2} (n \log n + m))$, dan dengan *additive error* maksimum $\epsilon \Delta G$ (ϵ adalah konstanta yang *fixed* dan ΔG adalah diameter dari graf G). Akan tetapi skor *betweenness centrality* lebih sulit untuk diestimasi dan kualitas pendekatan yang ada tergantung pada verteks sumber (*source vertex*). Brandes dan Pich [7] mengusulkan estimasi *betweenness* yang bersifat heuristik dengan strategi berbeda untuk menentukan *source vertex.* Mereka menemukan bahwa pemilihan secara acak dari *source vertex* jauh lebih baik dibandingkan dengan strategi deterministik. Sementara itu, belum ada algoritma untuk menghitung *centrality* untuk satu verteks yang lebih cepat dari menghitung *centrality* untuk semua verteks. Dalam paper ini dibandingkan algoritma eksak dan algoritma pendekatan. Algoritma pendekatan yang digunakan adalah yang dikembangkan oleh Bader et.al.[4]. berdasarkan sampling adaptif untuk menghitung nilai pendekatan *betweenness centrality* untuk verteks tertentu v.

Paper ini akan disusun sebagai berikut: bagian 2 berisi kajian pustaka, dilanjutkan dengan metode penelitian pada bagian 3, diikuti dengan diskusi pada 4 serta diakhiri dengan kesimpulan pada bagian 5.

2. Kajian Pustaka

Betweennes digunakan dalam menganalisis banyak masalah dunia nyata yang berhubungan dengan struktur dan analisis jaringan, seperti jaringan komunikasi, jejaring sosial, penyebaran penyakit dan lain-lain.

2.1 Solusi Eksak terhadap Betweenness Centrality

Sebelum memberikan algoritma Brandes untuk menghitung *betweenness centrality* terlebih dahulu diberikan pengamatan penting berikut yang dikenal dengan Bellman *criterion* [6].

Lemma 1. Sebuah verteks $v \in V$ terletak pada sebuah lintasan terpendek antara verteks s dan $t \in V$, jika dan hanya jika $d(s, t) = d(s, v) + d(v, t)$.

Diberikan pasangan jarak dan jumlah lintasan terpendek, *pair-dependency* $\delta_{st}(v) = \frac{\lambda st(v)}{\lambda st}$ dari pasangan $s, t \in V$ pada verteks antara $v \in V$, yakni rasio dari lintasan terpendek antara s dan t dimana v terletak di dalamnya, diberikan oleh:

$$\lambda_{st}(v) = \begin{cases} 0; & jika\ d(s,t) < d(s,v) + d(v,t) \\ \lambda sv.\lambda vt\ ; & jika\ tidak \end{cases} \quad (1)$$

Untuk mendapatkan *betweenness centrality* untuk verteks v, kita cukup menjumlahkan semua *pair-dependencies* untuk verteks tersebut sehingga diperoleh:

$$C_B(v) = \sum_{s \neq v \neq t \in V} \delta_{st}(v) \quad (2)$$

Dengan demikian, *betweenness centrality* dihitung dalam dua tahap yakni:

1. Hitung panjang dan jumlah lintasan terpendek antara semua pasangan verteks
2. Jumlahkan semua *pair-dependencies*

Kompleksitas menentukan *betweenness centrality* didominasi oleh langkah 2 algoritma di atas, yakni dengan waktu $\Theta(n^3)$ untuk perjumlahan dan ruang $\Theta(n^2)$ untuk menampung semua *pair-dependencies*. Situasi ini diperbaiki dengan lemma berikut ini.

Lemma 2 (*Algebraic path counting*) Misalkan $A^k = (a_{uv}^{(k)})_{u,v} \in V$ adalah pangkat k dari matriks *adjacency* dari sebuah graf tak berbobot, maka $a_{uv}^{(k)}$ sama dengan jumlah lintasan dari verteks u ke v dengan panjang = k.

Karena $O(n^2)$ *pair-dependencies* harus dijumlahkan untuk setiap verteks, waktu keseluruhan dari implementasi didominasi oleh waktu yang dibutuhkan dalam perkalian matriks. Pekerjaan ekstra dapat dihindarkan dalam contoh tertentu yang sesuai yang dikenal dengan *geodetic semiring* [5] yang merupakan generalisasi *semiring* tertutup untuk problema lintasan terpendek. Ini menghasilkan algoritma dengan kompleksitas $\Theta(n^3)$ untuk *betweenness* dengan memperluas algoritma Floyd-Warshall untuk problema lintasan terpendek dari semua pasangan dengan menghitung jumlah lintasan. Untuk menghitung jumlah lintasan terpendek, kita menggunakan algoritma Breadth-First Search (BFS) untuk graf tak berbobot dan algoritma Dijkstra untuk graf berbobot. Kedua algoritma diset dengan verteks awal $s \in V$ dan pada setiap tahap tambahkan verteks terdekat ke dalam verteks yang sudah dijalani hingga diperoleh lintasan terpendek ke semua verteks lain. Dalam proses ini, secara alami akan ditemukan semua lintasan dari verteks awal. Defenisikan himpunan *predecessor* dari sebuah verteks v pada lintasan terpendek dari s dengan:

$$P_s(v) = \{u \in V: \{u, v\} \in E, d(s,v) = d(s,u) + w(u,v)\} \quad (3)$$

Lemma 3 (*Combinatorial Shortes-Path Counting*) Untuk $s \neq v \in V$ maka:

$$\lambda_{sv} = \sum_{u \in Ps(v)} \lambda su \quad (4)$$

Bukti: Karena semua bobot edge adalah bilangan positif, edge terakhir dari sembarang lintasan terpendek dari s ke v adalah edge $\{u,v\} \in E$ sedemikian hingga $d(s,u) < d(s,v)$. Jelas bahwa jumlah lintasan terpendek dari s ke v yang berakhir dengan edge tersebut sama dengan jumlah lintasan terpendek dari s ke u. Kesamaan ini diperoleh sebagai konsekuensi dari Lemma 1. Algoritma BFS dan Dijkstra dengan mudah dapat dikembangkan untuk menghitung jumlah

lintasan terpendek sesuai dengan Lemma 3 di atas. BFS menghabiskan waktu sebesar O(*m*) dan algoritma Dijkstra membutuhkan waktu sebesar O(*m+nlogn*) jika *priority queue* diimplementasikan dengan *fibonacci heap*.

Konsekuensinya dinyatakan sebagai berikut: Berikan sebuah verteks sumber (awal) $s \in V$, maka panjang dan jumlah dari semua lintasan terpendek ke verteks lain dalam graf *G* dapat dihitung dalam waktu O(*m*) untuk graf tak berbobot dan O(*m + n log n*) untuk graf berbobot. Sebagai akibatnya, maka $\lambda_{sv,}\ s,v \in V$ dapat dihitung dalam waktu O(*nm*) untuk graf tak berbobot dan O(*nm + n^2 log n*) untuk graf berbobot. Dengan demikian waktu eksekusi didominasi pekerjaan untuk menjumlahkan *pair-dependencies* dengan kompleksitas $\Theta(n^3)$. Pada bagian berikut ini ditunjukkan bagaimana cara untuk mereduksi kompleksitas tersebut secara substansial dengan mengakumulasikan jumlahan parsial dari *pair-dependency*.[6][7]

Untuk menghilangkan keharusan menjumlahkan secara eksplisit semua *pair-dependency*, kita memperkenalkan ide ketergantungan (*dependency*) dari sebuah verteks $s \in V$ pada sebuah verteks $v \in V$ yang didefenisikan dengan: $\delta_{s\bullet}(v) = \sum_{t \in V} \delta st(v)$

Observasi penting dari rumus di atas adalah bahwa jumlahan parsial ini memenuhi sebuah relasi yang rekursif. Berikut ini adalah kasus khusus dari relasi tersebut yang dengan mudah dapat dikenali.

Lemma 5. Jika terdapat dengan pasti sebuah lintasan terpendek dari $s \in V$ ke setiap $t \in V$, *dependencies* dari *s* pada sembarang $s \in V$ memenuhi: $\delta_{s\bullet}(v) = \sum_{w:v \in Ps(w)} (1 + \delta s \bullet (w))$

Bukti: Asumsi yang diberikan menghasilkan bahwa verteks dan edge dari semua lintasan terpendek dari s akan membentuk sebuah pohon. Oleh karena itu, *v* akan terletak pada semua atau satu lintasan antara s dengan sembarang $t \in V$, yakni $\delta_{st}(v)$ sama dengan 1 atau 0. Lebih jauh, *v* terletak semua lintasan terpendek ke verteks dimana *v* adalah predesornya, dan pada semua lintasan terpendek yang terdapat pada *v*.

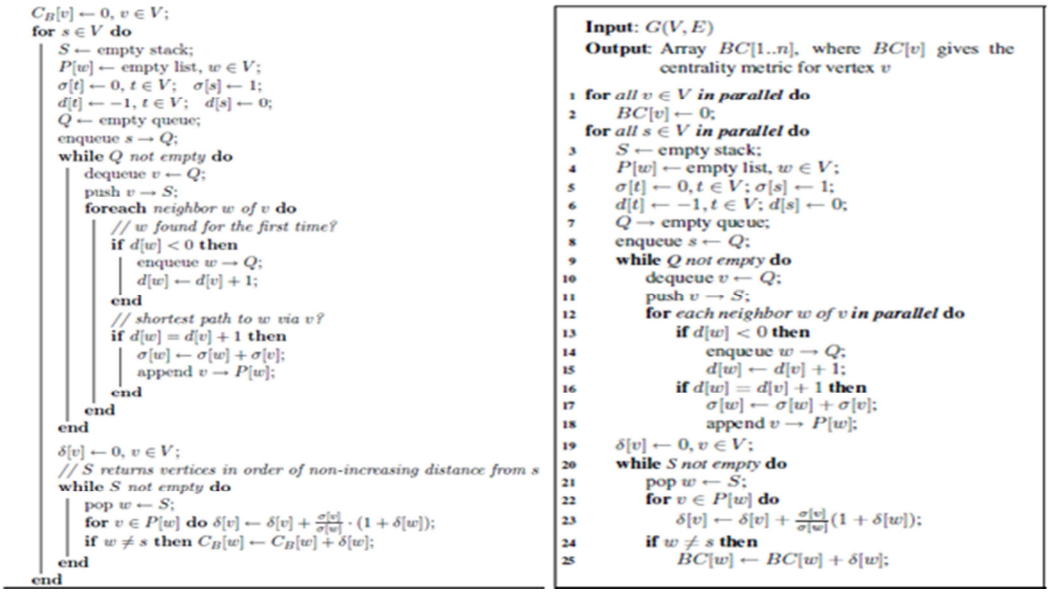
Algoritma eksak untuk menghitung *betweenness centrality* dari Brandes mempunyai kompleksitas waktu O(*nm*) untuk graf tak berbobot [6]. Sementara Algoritma paralel, yang dikembangkan Badder & Madduri [2] dimaksudkan sebagai pengembangan dari algoritma Brandes [4]. Dengan demikian diharapkan waktu eksekusi dapat direduksi menjadi O(*nm/p*) untuk graf tak berbobot serta O([*nm + n^2 log n*]/*p*) untuk graf berbobot dimana *p* adalah jumlah prosesor yang diparalelkan. Paralelisme dapat diterapkan dalam dua level yakni:

1. Perhitungan BFS/SSSP dari setiap verteks dan dijalankan secara konkuren, dengan syarat jumlah centrality diremajakan secara otomatis
2. Pada saat melaksanakan BFS/SSSP sesungguhnya dapat juga dijalankan secara paralel

Pseudocode untuk algoritma perhitungan eksak *betweenness centrality* untuk stand alone dan parallel dapat dilihat pada Gambar 1 di bawah ini.

2.2 Solusi Pendekatan terhadap *Betweenness Centrality*

Teknik sampling adaptif diperkenalkan oleh Lipton dan Naughton [11] untuk mengestimasi ukuran *transitive closure* dari sebuah graf. Sebelumnya algoritma untuk mengestimasi *transitive closure* didasarkan pada sampling secara acak terhadap verteks sumber, dan menyelesaikan problema ketercapaian sumber-tunggal untuk verteks yang terpilih, kemudian menggunakan informasi tersebut untuk mengestimasi ukuran *transitive closure.* Hal tersebut didasarkan pada lemma 6 berikut ini.



```
C_B[v] ← 0, v ∈ V;
for s ∈ V do
    S ← empty stack;
    P[w] ← empty list, w ∈ V;
    σ[t] ← 0, t ∈ V;   σ[s] ← 1;
    d[t] ← −1, t ∈ V;   d[s] ← 0;
    Q ← empty queue;
    enqueue s → Q;
    while Q not empty do
        dequeue v ← Q;
        push v → S;
        foreach neighbor w of v do
            // w found for the first time?
            if d[w] < 0 then
                enqueue w → Q;
                d[w] ← d[v] + 1;
            end
            // shortest path to w via v?
            if d[w] = d[v] + 1 then
                σ[w] ← σ[w] + σ[v];
                append v → P[w];
            end
        end
    end
    δ[v] ← 0, v ∈ V;
    // S returns vertices in order of non-increasing distance from s
    while S not empty do
        pop w ← S;
        for v ∈ P[w] do δ[v] ← δ[v] + (σ[v]/σ[w]) · (1 + δ[w]);
        if w ≠ s then C_B[w] ← C_B[w] + δ[w];
    end
end
```

```
Input: G(V, E)
Output: Array BC[1..n], where BC[v] gives the
        centrality metric for vertex v
1  for all v ∈ V in parallel do
2      BC[v] ← 0;
   for all s ∈ V in parallel do
3      S ← empty stack;
4      P[w] ← empty list, w ∈ V;
5      σ[t] ← 0, t ∈ V; σ[s] ← 1;
6      d[t] ← −1, t ∈ V; d[s] ← 0;
7      Q → empty queue;
8      enqueue s ← Q;
9      while Q not empty do
10         dequeue v ← Q;
11         push v → S;
12         for each neighbor w of v in parallel do
13             if d[w] < 0 then
14                 enqueue w → Q;
15                 d[w] ← d[v] + 1;
16             if d[w] = d[v] + 1 then
17                 σ[w] ← σ[w] + σ[v];
18                 append v → P[w];
19     δ[v] ← 0, v ∈ V;
20     while S not empty do
21         pop w ← S;
22         for v ∈ P[w] do
23             δ[v] ← δ[v] + (σ[v]/σ[w])(1 + δ[w]);
24         if w ≠ s then
25             BC[w] ← BC[w] + δ[w];
```

(a) (b)

Gambar 1. *Pseudocode* Menghitung *Betweennes* (a) *stand alone* [6], (b) paralel [2]

Lemma 6. $C_B(v)$ sama dengan nol jika dan hanya jika tetangga verteks tersebut membentuk sebuah *clique.*

Misalkan a_i menyatakan *dependency* dari verteks v_i pada v, yakni $a_i = \delta_{vi\bullet}(v)$. Misalkan juga $A = \sum a_i = C_B(v)$. Dengan mudah dapat diverifikasi bahwa $0 \leq a_i \leq n-2$ dan $0 \leq A \leq (n-1)(n-2)/2$. Jumlah yang ingin kita estimasi adalah A. Hal tersebut dilakukan dengan algoritma 3 berikut ini.

Algoritma 3. Secara berulang-ulang lakukan sampling sebuah verteks $v_i \in V$; lakukan SSSP (menggunakan BFS atau Dijkstra) dari v_i dan pertahankan sebuah array S yang berisi skor *dependency* $\delta_{vi\bullet}(v)$. Lakukan sampling hingga S lebih besar dari cn untuk konstanta $c \geq 2$. Misalkan jumlah total dari sampel adalah k, maka estimasi skor *betweenness centrality* $C_B(v) = \frac{nS}{k}$.

Misalkan X_i adalah variabel acak yang merepresentasikan *dependency* dari verteks yang dipilih secara acak pada v. Kemudian untuk menganalisis algoritma ini, kita akan mengembangkan Lemma berikut.

Lemma 7. Misalkan $E[X_i]$ menyatakan ekspektasi dari X_i dan $Var[X_i]$ menyatakan variansi dari X_i, maka diperoleh $E[X_i] = A/n$, $E[X_i^2] \leq A$, dan $Var[X_i] \leq A$

Lemma 8 dan 9 berikut ini berguna untuk membuktikan bahwa batas bawah pada jumlah sampel yang diharapkan diperoleh sebelum loop berhenti.

Lemma 8. Misalkan $k = \epsilon n^2/A$, maka $\Pr[X_1 + X_2 + ... + X_k \geq cn] \leq \frac{\epsilon}{(c-\epsilon)\text{^}2}$

Lemma 9. Misalkan $k = \epsilon n^2/A$ dan $d > 0$, maka $\Pr[|\frac{n}{k}(\sum_{i=1}^{k} Xi) - A| \geq dA] \leq 1/\epsilon d^2$

Teorema 1. Misalkan A adalah estimasi untuk A pada prosedur di atas dan misalkan $A > 0$, maka untuk $0 < \epsilon < 0.5$ dengan probabilitas lebih besar atau sama dengan $1 - 2\epsilon$, algoritma 3 memberikan estimasi A ke dalam faktor $1/\epsilon$

Bukti: Terdapat dua cara yang memungkinkan algoritma 3 gagal: (i) dapat berhenti telalu cepat untuk menjamin sebuah batas error yang baik, atau (ii) dapat berhenti setelah sampel yang didapat cukup dengan nilai estimasi yang buruk. Pertama kita menyatakan bahwa algoritma 3 hampir tidak mungkin berhenti dengan $k \leq n^2/A$. Kita memperoleh bahwa :

$\Pr[(\exists_j)(j \le k)\ \&\ (X_1 + X_2 + ... + X_k \ge cn)] \le \Pr[X_1 + X_2 + ... + X_k \ge cn]$ dimana $k = \epsilon n^2/A$, karena peristiwa di sebelah kanan pertidaksamaan mengimplikasikan peristiwa yang di sebelah kiri. Tetapi dengan lemma 8, bagian sebelah kanan dari persamaan ini maksimum $\epsilon/(c\text{-}\epsilon)^2$. Dengan mensubstitusikan $c = 2$ dan $0 < \epsilon < 0.5$, kita mendapatkan bahwa probabilitas di atas lebih kecil dari ϵ. Berikut ini dijelaskan akurasi dari estimasi. Jika $k = \epsilon n^2/A$, dengan Lemma 9 maka estimasi $\tilde{A} = \frac{n}{k}\sum_{i=1}^{k} Xi$ ada dalam jangkauan dA dari A dengan probabilitas $\ge 1/ (\epsilon d^2) = \epsilon$ jika dimisalkan $d = 1/\epsilon$. Dengan menggabungkan kedua kemungkinan gagal di atas, kita peroleh bahwa probabilitas total kegagalan lebih kecil dari $\epsilon + (1 - \epsilon)\,\epsilon < 2\epsilon$. Akhirnya, jika A > 0, maka terdapat paling tidak satu i sedemikian hingga $a_i > 0$, sehingga algoritma berhenti. Aspek penting dari teorema 1 adalah proses sampling bersifat adaptif dan biasanya melakukan sampling untuk jumlah sampel yang stabil. Dengan mensubstitusikan $A = n^2/t$ pada analisis ini, kita memperoleh teorema berikut:

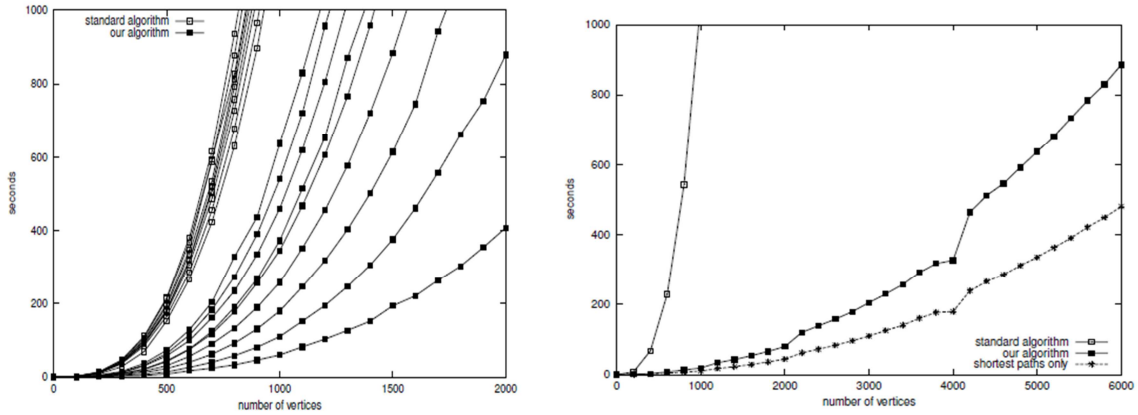
Teorema 2. Untuk $0 < \epsilon < 0.5$, jika *centrality* dari sebuah verteks $v = n^2/t$ untuk sembarang konstanta t, maka dengan probabilitas $\ge 1 - 2\epsilon$ *centrality* verteks v dapat diestimasi sekitar faktor $1/\epsilon$ dengan ϵt buah sampel verteks sumber. [4]

3. Metode Penelitian

Penelitian ini dilakukan dengan membandingkan perhitungan *betweenness centralility* dengan menggunakan algoritma Brandes baik untuk *stand alone* maupun paralel dan algoritma pendekatan berdasarkan sampling adaptif.

3.1 Pengujian Algoritma Brandes *Stand Alone*

Pengujian dilakukan terhadap graf berarah dan tidak berarah menggunakan *the Library of Efficient Data Structures and Algorithms* (LEDA, dari Mehlhorn & Näher [13]) seperti terdapat pada [6]. Eksperiman dilakukan dengan menggunakan prosesor tunggal SparcStation Sun Ultra 10 dengan kecepatan CPU 440 MHz dan ukuran memori 256 Mbytes. Pengujian pertama dilakukan pada graf acak dengan jumlah verteks antara 100 sampai 2000 dengan densitas (didefenisikan sebagai m /[$(n(n-1)/2$] antara 10% sampai 90%. Pengujian kedua dilakukan terhadap graf tak berarah dan tak berbobot dengan 494 aktor dan 1774 hubungan (edge), algoritma Brandes membutuhkan waktu 448 detik dengan menggunakan memori kurang dari 8 Mbytes. Hasil pengujian pertama dan kedua ditunjukkan pada Gambar 2 di bawah ini. [6]



Gambar 2. Hasil Pengujian Algoritma Brandes

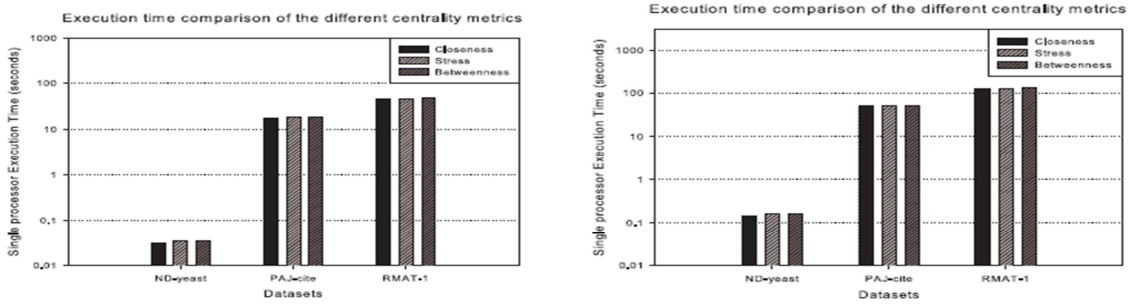
3.2 Pengujian Algoritma Brandes Paralel

Pengujian terhadap algoritma ini seperti terdapat pada [2] dilakukan terhadap data set dengan karakteristik seperti terlihat pada Tabel 1 di bawah ini.

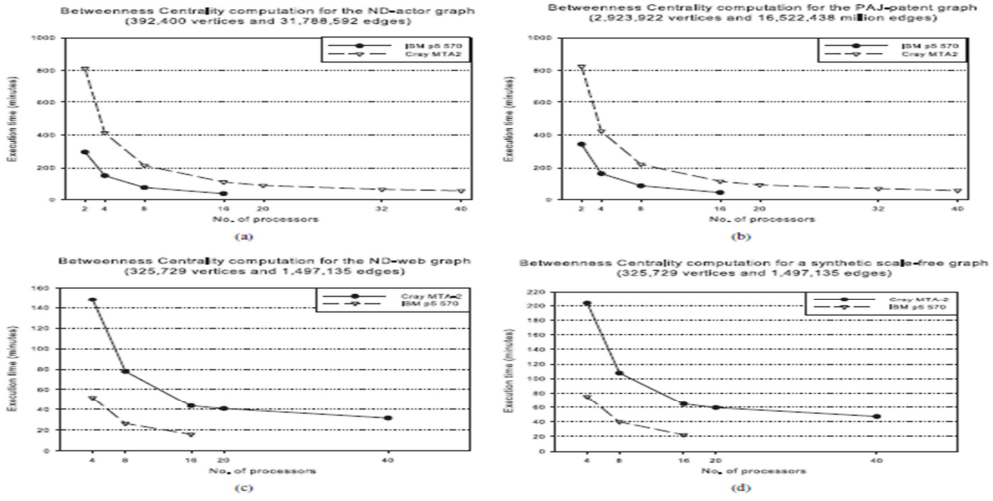
Tabel 1. Karakteristik Data Set yang Diuji [2]

Dataset	Deskripsi Jaringan
ND-Actor	Sebuah graf tak berarah dengan jumlah verteks = 392.400 (aktor film) dan jumlah edge = 31.788.592. Sebuah edge menyatakan link antara 2 aktor, jika ada link antara kedua aktor berarti mereka pernah main dalam film yang sama. Data set juga berisi daftar aktor untuk 127.823 film
ND-Web	Sebuah graf berarah dengan 325.729 buah verteks dan 1.497.135 edge (27.455 buah loop). Vertkes menyatakan web page dan link menyatakan garis dari - ke
ND-Yeast	Sebuah graf tak berarah dengan 2.114 buah verteks dan 2.277 edge (74 loop). Verteks menyatakan protein dan edge merupakan interaksi antar protein dalam jaringan yeast
PAJ-Patent	Sebuah jaringan dengan kira-kira 3 juta paten yang diberikan antara Januari 1963 sampai Desember 1999, dan 16 juta sitasi sesama mereka antara 1975 sampai 1999
PAJ-Cite	Dataset sitasi Lederberg dengan 8.843 buah verteks dan 41.609 buah edge dalam PAJEK

Kemudian dihasilkan sebuah graf sintesis dengan ukuran sama dengan ND-Actor yang diperoleh dengan menjalankan paket R-MAT sebagai perbandingan. Data set tersebut diujikan dengan prosesor tunggal jenis IBM p5 570 dan Cray MTA-2. Kemudian dilakukan pengujian secara paralel dengan jenis prosesor sama dan dengan level paralelisme sampai 40 CPU untuk Cray MTA-2. Hasilnya ditunjukkan berturut-turut pada Gambar 3 dan Gambar 4 di bawah ini.



Gambar 3. Perbandingan waktu eksekusi perhitungan *centrality* dengan prosesor berbeda



Gambar 4. Perhitungan *betweennes* dengan multiprosesor berbeda untuk graf berbeda

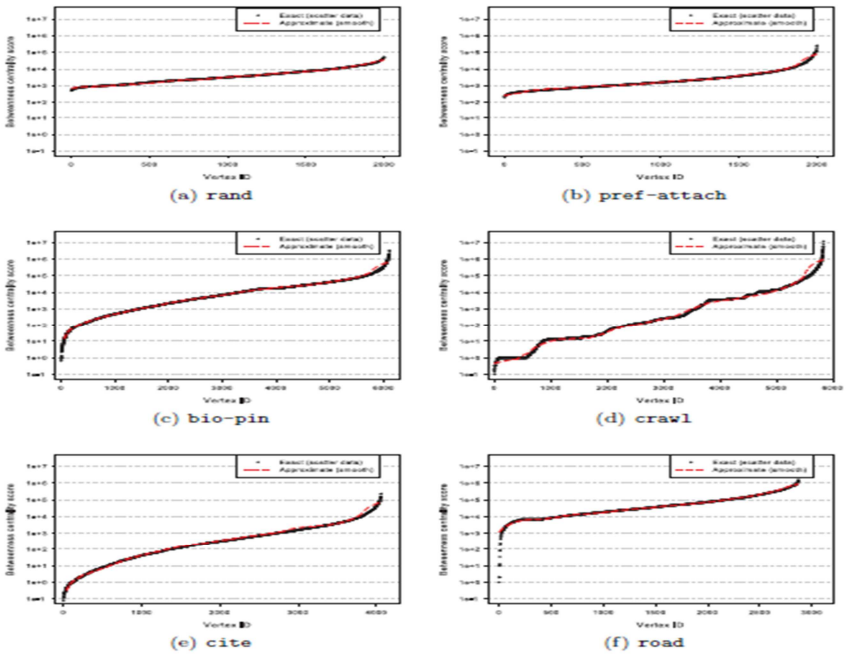
3.3 Pengujian Algoritma Pendekatan Sampling Adaptif

Pengujian terhadap algoritma pendekatan ini telah dilakukan terhadap data set seperti terlihat pada Tabel 2 berikut ini.

Tabel 2. Jaringan yang Digunakan untuk Pengujian Pendekatan Sampling Adaptif [4]

Label	Jaringan	n	m	Rincian
Rand	Graf Acak	2000	7980	Sintetis, tak berarah
Pref-attach	Preferential Attachment	2000	7980	Sintetis, tak berarah
Bio-pin	Human Protein Interaction	8503	32191	Tak berarah
Crawl	Web-Crawl (standford.edu)	9914	36854	Berarah
Cite	Lederberg Citation Network	8843	41001	Berarah
Road	Rome, Italy Road Network	3353	4435	Berbobot, tak berarah

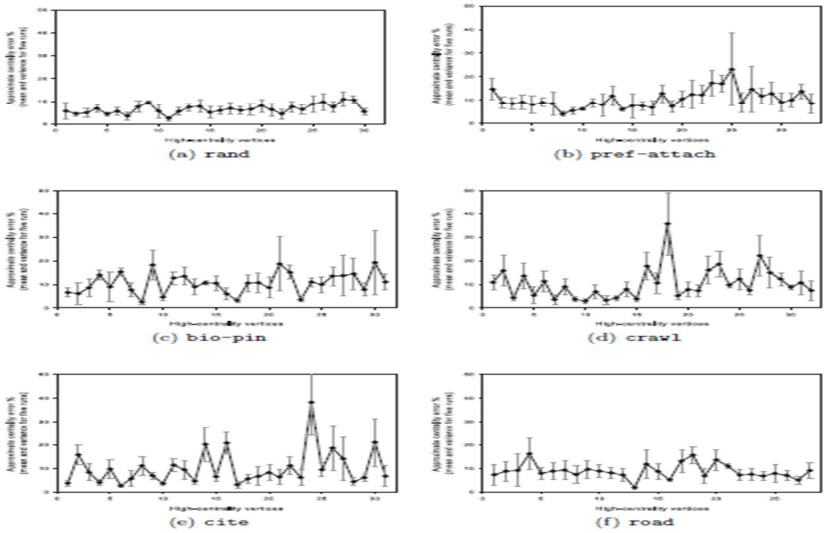
Pengujian 1: pengujian ini dimaksudkan untuk membandingkan antara solusi eksak dengan solusi pendekatan (estimasi) untuk keenam graf ada Tabel 2 di atas. Hasil pengujian menunjukkan bahwa jaringan **rand** dan **pref-attach** mempunyai peningkatan waktu yang lebih lambat dibandingkan dengan ke-4 jaringan nyata. Juga ditunjukkan bahwa terdapat suatu persentasi verteks dengan *low-centrality* (skor lebih kecil atau mendekati n) yang signifikan pada jaringan **cite**, **crawl** dan **bio-pin**. Algoritma pendekatan digunakan untuk mengestimasi nilai *betweenness centrality* untuk semua verteks dari jaringan yang diuji. Untuk memberikan visualisasi yang memadai maka dilakukan penghalusan kurva pada estimasi *betweenness centrality*. Untuk pengujian ini diset parameter c = 5 dan mengurangi jumlah sampel hingga $\frac{n}{20}$, seperti terlihat pada Gambar 5 di bawah ini.



Gambar 5. Perbandingan nilai *betweenness* eksak dengan nilai pendekatan

Pengujian 2: pengujian ini dimaksudkan untuk mengukur kualitas dari algoritma pendekatan untuk graf dengan verteks yang *high-centrality* dengan memilih secara acak sekitar 1% dari verteks yang ada. Kemudian dari verteks yang terpilih dihitung estimasi nilai *centrality* untuk

setiap jaringan yang diuji. Pengujian dilakukan sebanyak 5 kali untuk setiap verteks dengan nilai c = 5. Verteks diurutkan dari kiri ke kanan berdasarkan nilai *centrality* eksak pada sumbu mendatar dan persentase error diplot pada sumbu tegak. Hasil persentase error untuk rata-rata dan varians diperlihatkan pada Gambar 6 di bawah ini.



Gambar 6. Persentase rata-rata error estimasi *betweennes* untuk 5 kali eksekusi

4. Diskusi

4.1 Analisis Algoritma Brandes (Algoritma Eksak)

Gambar 2 di atas menunjukkan bahwa algoritma Brandes dapat mencapai kecepatan jauh melebihi kecepatan algoritma standar. Kemudian algoritma Brandes paralel akan memberikan waktu eksekusi yang semakin baik. Gambar 3 menunjukkan bahwa ketiga metrik (*closeness*, *stress* dan *betweenness*) untuk 3 jenis data set mempunyai kompleksitas waktu yang hampir sama dan waktu eksekusi nyata yang tidak jauh berbeda. Gambar 4 (a) memberikan waktu eksekusi terhadap graf ND-Actor dan hasilnya menunjukkan pola linier terhadap jumlah prosesor, dengan waktu eksekusi 42 menit dengan 16 buah prosesor IBM p5 570. Hasil yang hampir sama diperoleh untuk graf PAJ-Patent, tetapi waktu eksekusi sangat tergantung pada ukuran keterhubungan dari graf yang diuji. Gambar (c) dan (d) memberikan hasil waktu eksekusi pada prosesor MTA-2 dan p5 570 untuk graf ND-Web dan sebuah graf sintesis dengan ukuran sama. Untuk ukuran graf sama, graf sintesis membutuhkan waktu jauh lebih lama dibandingkan dengan ND-Web. Waktu eksekusi menurun secara signifikan jika jumlah prosesor = 40 unit jenis Cray MTA-2. Dalam hal ini masih perlu dipikirkan bagaimana cara untuk penanganan otomatis terhadap paralel yang bersarang (*nested parallelism*)

4.2 Analisis Algoritma Pendekatan

Dari pengujian pertama, algoritma pendekatan dibandingkan dengan algoritma eksak dan hasilnya didapat bahwa nilai estimasi *betweenness centrality* hampir berimpit dengan nilai eksak seperti terlihat pada Gambar 5. Pengujian kedua memberikan persentase error rata-rata dan variansi untuk 5 kali eksekusi menunjukkan bahwa algoritma pendekatan ini memberikan kinerja yang cukup baik yang tidak jauh berbeda dengan hasil yang diberikan oleh algoritma eksak.

5. Kesimpulan

Dalam paper ini telah dilakukan kajian terhadap algoritma perhitungan *betweenness centrality* secara eksak dan pendekatan menggunakan teknik sampling adaptif. Sebagai kesimpulan dicatat beberapa temuan yakni: (1) kompleksitas waktu algoritma eksak perhitungan *betweenness centrality* adalah O(*nm*) untuk graf tak berbobot dan O($nm+n^2 \log n$) untuk graf berbobot; (2) kompleksitas ruang untuk perhitungan *betweenness centrality* adalah O($n+m$); (3) waktu eksekusi dapat ditingkatkan dengan menggunakan konsep komputasi paralel; (4) algoritma pendekatan sampling adaptif memberikan waktu yang hampir sama dengan algoritma eksak; (5) pengembangan algoritma yang bersifat dinamis dengan kompleksitas waktu lebih baik sangat dibutuhkan untuk menganalisis jaringan yang semakin besar dan dinamis saat ini.

Referensi

[1] Anthonisse, J., 1971, *The rush in a directed graph*, Report BN9/71, Stichting Mathematisch Centrum, Amsterdam

[2] Bader, D., Madduri, K., August 2006, *Parallel algorithms for evaluating centrality indices in real-world networks*, Proc. 35th Int'l Conf. on Parallel Processing (ICPP), Columbus, OH, IEEE Computer Society

[3] Bader, D., Madduri, K., 2007,*Small-world Network Analysis in Parallel (SNAP)*: a toolkit for centrality analysis. http://www.cc.gatech.edu/~kamesh

[4] Bader, D. et.al., December 2007, *Approximating betweenness centrality*, Proc. 5th Int'l. Workshop on Algorithms and Models for the Web-Graph, Vol 4863 of LNCS, 124-137.

[5] Batagelj, V., 1994, *Semirings for social network analysis*, J. Mathematical Sociology, Vol. 19, No.1, 53-68

[6] Brandes, U., 2001, *A faster algorithm for betweenness centrality*, J. Mathematical Sociology, Vol. 25, No. 2, 163-177

[7] Brandes, U., Pich, C., 2007, *Centrality estimation in large networks*, Intl. Journal of Bifurcation and Chaos, Special Issue on Complex Networks' Structure and Dynamics

[8] Eppstein, D., Wang, J., 2001, *Fast approximation of centrality*, Proc. 12th Ann. Symp. Discrete Algorithms (SODA-01), Washington, DC., 228-229

[9] Freeman, L. C., 1977, *A set of measures of centrality based on betweenness*, Sociometry, Vol. 40, No. 1 , 35-41

[10] Girvan, M., Newman, M., 2002, *Community structure in social and biological networks*, Proceedings of the National Academy of Sciences USA, Vol. 99, No. 12, 7821-7826

[11] Lipton, R., Naughton, J., 1989, *Estimating the size of generalized transitive closures*, VLDB., 165-171

[12] Mehlhorn, K. and Näher, S., 1999, *The LEDA Platform of Combinatorial and Geometric Computing*, Cambridge University Press, home page at http://www.mpi-sb.mpg.de/LEDA/

[13] Newman, M., 2003, *The structure and function of complex networks*, SIAM Review, Vol. 45, No. 2, 167-256