Perbandingan Implementasi Algoritma-algoritma Pagerank pada Satu Mesin Komputer

Farhan Herdian Pradana[1], Muhammad Eka Suryana[2], Med Irzal[3]

Program Studi Ilmu Komputer, Fakultas Matematika dan Ilmu Pengetahuan Alam
Universitas Negeri Jakarta, DKI Jakarta, Indonesia
[1]farhan.herdia123@gmail.com, [2]eka-suryana@unj.ac.id, [3]medirzal@unj.ac.id

Abstrak

Algoritma Pagerank merupakan algoritma mengurutkan halaman web pada search engine Google. Masalah pada Algoritma Pagerank adalah memerlukan memori utama yang cukup besar, dan tidak mungkin dilakukan pada satu mesin komputer dengan memori utama yang terbatas. Akan dicari algoritma alternatif dari Algoritma Pagerank Original buatan Google dengan membandingkannya pada algoritma-algoritma Pagerank dari penelitian lainnya dengan membandingkan kecepatan, penggunaan memori utama, dan kemiripan hasil. Penelitian dilakukan dengan melakukan pengkodean terhadap algoritma Pagerank Original, algoritma Distributed Pagerank Computation (DPC), algoritma Modified DPC (MDPC), dan algoritma Random Walker. Semua kode program dijalankan pada dataset dan dibandingkan kecepatan, penggunaan memori utama, dan kemiripan hasil akhirnya. Khusus hasil akhir, hasil dari algoritma Random Walker dijadikan acuan karena dasar dari Algoritma Pagerank adalah Random Walker. Hasilnya Algoritma Pagerank Original unggul dari sisi kecepatan dan hasil yang mirip dengan hasil Algoritma Random Walker. Sedangkan algoritma DPC dan MDPC unggul di penggunaan memori utama yang lebih hemat, sehingga cocok untuk satu mesin komputer yang memiliki memori utama yang terbatas, tetapi dengan catatan mengorbankan kecepatan yang lebih lambat dan hasil yang tidak mirip terhadap Random Walker.

I. PENDAHULUAN

Internet adalah jaringan luas yang membuat jaringan komputer seluruh dunia dapat saling berkomunikasi (Sample, 2018). Internet melahirkan World Wide Web (WWW) yang memungkinkan situs web atau biasa disebut *website* untuk bisa diakses oleh semua orang. *Website* adalah sekumpulan halaman web yang saling terkait dan berada pada nama domain yang sama. Website dapat dibuat dan dipelihara oleh seorang individu, grup, perusahaan, atau organisasi lain dengan berbagai macam tujuan (Techopedia, 2020). Pada tahun 1992 hanya terdapat sepuluh *website*, lalu pada tahun 1994 angka ini bertumbuh menjadi 3.000 *website*, dan semakin bertumbuh pesat pada tahun 2021 menjadi kurang lebih 1,88 miliar *website* (Amstrong, 2021).

Dengan meledaknya jumlah halaman web, memunculkan tantangan baru dalam memperoleh informasi dari web. Pengguna biasanya menelusuri web dengan mengunjungi graf *link* yang terdapat pada halaman web, biasanya dimulai pada situs kumpulan index halaman web berkualitas tinggi yang dipelihara oleh manusia seperti Yahoo.com, atau menggunakan *search engine* (Brin dan Page, 1998). Seiring perkembangan zaman, *search engine* Google menjadi *search engine* teratas dengan pengguna terbe-

sar di dunia dengan penguasaan pasar sebesar 91% (GlobalStatCounter, 2022). Kunci kesuksesan dari Google terletak pada Pagerank. Pagerank merangking halaman web berdasarkan kepentingan relatif (*relative importance*) suatu halaman web berdasarkan graf tautan web (Page dkk., 1999).

Dalam melakukan perangkingan halaman web, Pagerank dapat didefinisikan pada persamaan 1. Dimana u adalah halaman web. F_u adalah himpunan halaman u yang menunjuk halaman lain dan B_u adalah himpunan halaman yang menunjuk ke u. $C_u = |F_u|$ adalah jumlah *link* dari u dan c adalah faktor yang digunakan untuk normalisasi (sehingga jumlah total *ranking* semua halaman web adalah konstan) dan $c < 1$. $E(u)$ adalah vektor yang berkorespondensi dengan *ranking* halaman web. $||\pi'||_1 = 1$. Iterasi perhitungan terus dilakukan sampai konvergen. Jika diubah kedalam persamaan matriks, maka persamaan 1 dapat diubah menjadi persamaan 2. Dimana X adalah matriks persegi yang baris dan kolomnya berkorespondensi dengan halaman web, dengan elemen $X_{u,v} = \frac{1}{C_u}$ jika terdapat *link* pada halaman u ke halaman v atau $X_{u,v} = 0$ jika tidak ada.

$$\pi'(u) = c \sum_{v \in B_u} \frac{\pi'(v)}{C_v} + cE(u) \qquad (1)$$

$$\pi' = c(X\pi' + E) \qquad (2)$$

Dasar intuitif dari persamaan 1 adalah *random walks* pada sebuah graf. Anggap pengguna internet sebagai *"random surfer"* yang terus meng-klik *link* selanjutnya secara acak. Namun, jika pengguna terjebak pada sebuah lingkaran halaman web (*link* yang diklik terus menampilkan halaman web yang pernah dikunjungi sebelumnya), tidak mungkin pengguna akan terus mengikuti *link* tersebut, melainkan pengguna akan langsung pindah ke halaman lain. Oleh sebab itu faktor E dipakai untuk memodelkan perilaku ini (Pengguna akan bosan dan langsung lompat ke halaman web lain berdasarkan distribusi pada E) (Page dkk., 1999). E dapat didefinisikan oleh pengguna (*user-defined parameter*) dan nilai elemennya dapat diisi dengan nilai yang seragam atau berbeda-beda. Menariknya, jika nilai elemen E dibuat berbeda-beda, maka dapat membuat Pagerank yang dipersonalisasi (Page dkk., 1999).

Walaupun persamaan Pagerank terlihat sederhana, terdapat masalah dari sisi ruang dan waktu. Dari sisi ruang, misal terdapat 1000 halaman web, maka akan terbentuk 1000x1000 matriks X. Misal tiap sel elemen X memerlukan memori 8 *Byte*, maka untuk membentuk 1000x1000 matriks X, tanpa menghitung alokasi *overhead* memori, memerlukan 8 *Mega Byte* (MB) memori utama (Lihat tabel 1). Di internet terdapat miliaran *website* dan setiap *website* dapat memiliki lebih dari 1 halaman, sehingga untuk bisa membentuk matriks X membutuhkan memori utama dengan kapasitas mencapai *Peta Byte* atau bahkan *Exa Byte*. Hal tersebut sangat tidak mungkin dilakukan pada komputer pribadi biasa yang memori utamanya hanya pada kisaran 4 GB sampai 32 GB, yang berarti ketika program dieksekusi langsung *crash* karena memori yang tidak cukup. Dari sisi waktu, proses *string matching* untuk mengakses nilai *ranking* suatu halaman web berdasarkan *string* URLnya juga memiliki kompleksitas waktu yang besar yakni O(NM), jika dilakukan dengan cara dicari satu-persatu. Beruntung *database* seperti MySQL menggunakan B-Tree dalam mengindeks datanya (MySQL-Doc, 2022). B-tree memiliki kompleksitas waktu kecil yakni hanya O(log(n)) (GeeksForGeeks, 2022).

Telah dilakukan penelitian tentang Pagerank yang terdistribusi dengan metode *iterative aggregation-disaggregation* (IAD) dengan *Block Jacobi smoothing* (Zhu dkk., 2005). Sederhananya, dilakukan *divide-and-conquer* dengan mengelompokan halaman web

Tabel 1: *Tabel alokasi memori utama untuk membentuk matriks X*

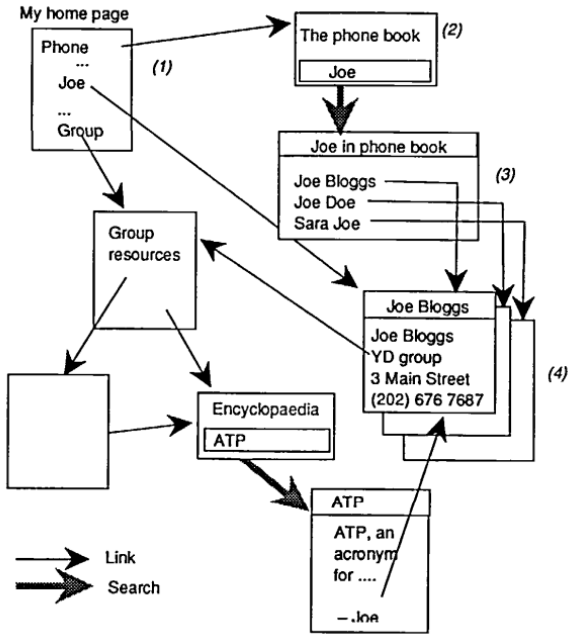
Jumlah Halaman Web	Memori (*Byte*)
256	524416
512	2097280
1024	8388736
2048	33554560
4096	134217856
8192	536871040
16384	2147483776
32768	8589934720

berdasarkan *domain*-nya lalu dihitung Pagerank lokalnya dan disatukan dengan metode komunikasi yang hemat memori dengan sebuah koordinator (Zhu dkk., 2005). Hasilnya, ditemukan sebuah metode Pagerank terdistribusi yang bisa dijalankan pada memori utama kecil dan lebih cepat konvergen sehingga menghemat waktu (Zhu dkk., 2005). Oleh karena itu, akan dicari algoritma Pagerank alternatif yang dapat dijalankan pada satu mesin komputer dengan memori utama terbatas, dengan cara melakukan perbandingan implementasi beberapa algoritma Pagerank pada satu mesin komputer.

II. Kajian Teori

World Wide Web (WWW)

WWW merupakan proyek Tim Berner-lee bersama teman-temannya, yang ditunjukan pada publik pada tahun 1991. WWW didesain untuk membawa sebuah semesta informasi global menggunakan teknologi yang ada. Dengan adanya WWW manusia dapat mengakses seluruh informasi melalui sebuah *platform browsing* apapun. Pada masa itu, sudah ada teknologi serupa yang membuat WWW mungkin untuk dilakukan. Sistem *hypertext* yang sudah ada pada saat itu, terbatas pada sistem *file* lokal atau terdistribusi dan kadang hanya dikembangkan

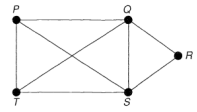


Gambar 1: *Gambar sebuah web yang terdiri atas kumpulan link dan indeks (Berners-Lee dkk., 1992)*

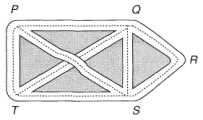
pada *platform* tertentu. Selain itu, juga ada sistem pengambilan dan akses informasi seperti Alex, Gopher, Prospero, dan WAIS yang sudah mencakup area yang luas, tetapi tanpa fungsionalitas *hypertext*. WWW menggabungkan teknik *hypertext*, *information retrieval*, dan *wide area networking* (Berners-Lee dkk., 1992).

Model yang dipakai WWW menggunakan dua paradigma dari *hypertext link* dan pencarian teks yang saling melengkapi. Gambar 1 menunjukkan bagaimana sebuah web yang berisi informasi pribadi terbentuk pada paradigma ini. Pembaca mulai pada halaman *home* (1) lalu menggunakan *link* grup atau publik untuk mencari bahan informasi. Indeks seperti buku telepon (2) ditampilkan sebagai dokumen yang memungkinkan untuk melakukan input pencarian. Hasil dari pencarian berupa dokumen *hypertext* virtual (3) yang menunjuk pada dokumen yang ditemukan (4) (Berners-Lee dkk., 1992).

Teori Graf

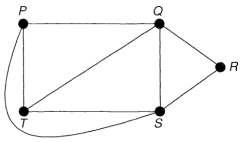


Gambar 2: *Contoh graf (Wilson, 1996)*

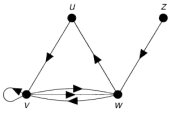


Gambar 3: *Contoh peta jalan yang dapat diandaikan sebagai graf (Wilson, 1996)*

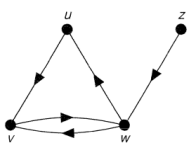
Graf adalah sebuah representasi dari himpunan titik (*node / vertice*) dan bagaimana titik-titik tersebut saling terhubung tanpa memperdulikan sifat metriknya (Wilson, 1996). Pada gambar 2 merupakan contoh graf, dengan P, Q, R, S, T merupakan titik, dan masing-masing terhubung dengan garis (*edge*). Garis yang menghubungkan titik P dan S disebut dengan PS, sedangkan garis yang menghubungkan titik Q dan T disebut dengan QT. Persilangan antara PS dan QT tidak disebut sebagai titik, karena keduanya tidak saling bersilangan, melainkan saling melompati layaknya gambar 3. Selanjutnya, kedua graf dianggap sama, jika dan hanya jika titik yang berkorespondensi sama-sama terhubung dengan garis yang sama dengan garis pada graf lainnya (Wilson, 1996). Sebagai contoh, graf pada gambar 4 merupakan graf yang sama dengan graf pada gambar 2 (Wilson, 1996).



Gambar 4



Gambar 5: *Contoh digraf (Wilson, 1996)*



Gambar 6: *Contoh digraf sederhana (Wilson, 1996)*

Garis pada graf dapat diberikan arah. Garis pada graf yang berarah disebut sebagai busur (*arc*). Graf yang memiliki arah pada garisnya disebut dengan graf berarah (*directed graph / digraph /* digraf) yang terdiri atas himpunan titik dan busur (Wilson, 1996). Pada digraf di gambar 5 terdapat himpunan titik u, v, w, dan z, dengan busur uv, vv, $vw(2\times)$, wv, wu, dan zw. Sebuah digraf disebut sebagai digraf sederhana jika himpunan busurnya tidak ada yang sama (*distinct*) dan tidak memiliki *loop* (contoh: busur vv) (Wilson, 1996). Digraf pada gambar 6 adalah contoh digraf sederhana.

Pagerank

Jika World Wide Web diibaratkan pada sebuah graf berarah, halaman web adalah titik graf, *link* adalah garis. Lalu *link* yang menunjuk keluar dari titik disebut *forward link*, sedangkan *link* yang menunjuk kedalam titik disebut *backlink*. Sangat sulit untuk mengetahui semua *backlink* suatu halaman web, tetapi sangat mudah untuk mengetahui semua *forward link* suatu halaman web yaitu dengan cara mengunduh halaman web tersebut (Page dkk., 1999). Setiap halaman web memiliki jumlah *backlink* yang beragam. Pada saat Pagerank diteliti, halaman *home* NetScape memiliki 62.804 *backlink* dibandingkan halaman web kebanyakan yang hanya memiliki beberapa *backlink*. Secara umum suatu halaman web jika memiliki banyak *backlink* dapat dianggap lebih penting daripada halaman web lain yang memiliki lebih sedikit *backlink*. Perhitungan jumlah sitasi sederhana pernah digunakan untuk memprediksi pemenang Nobel masa depan (Page dkk., 1999).

$$P_{ij} = \begin{cases} \frac{d}{C_j} + \frac{(1-d)}{N} & j \to i \\ \frac{(1-d)}{N} & j \not\to i \text{ dan } C_j \neq 0 \\ \frac{1}{N} & C_j = 0 \end{cases} \quad (3)$$

Graf pada WWW, dapat direpresentasikan

sebagai matriks transisi. Sebelumnya matriks transisi didefinisikan sebagai matriks X, berubah menjadi matriks P sesuai dengan penelitian Zhu dkk. (2005). Matriks P didefinisikan pada persamaan 3. C_j adalah jumlah *forward link* dari halaman j. $j \rightarrow i$ adalah halaman j memiliki *link* menuju halaman i.

Selanjutnya algoritma Pagerank didefinisikan pada algoritma 1

Algoritma 1 Pagerank (Zhu dkk., 2005)

1: Definisikan π^0 awal-awal
2: $k \leftarrow 0$
3: $\pi^{\sim k+1} \leftarrow P\pi^k$
4: $\pi^{k+1} \leftarrow \frac{\pi^{\sim k+1}}{||\pi^{\sim k+1}||_1}$
5: Jika $||\pi^{k+1} - \pi^k|| < \epsilon$ berhenti dan kembalikan nilai π^{k+1}
6: $k \leftarrow k + 1$
7: Ulangi langkah 3

Distributed Pagerank Computation (DPC)

Jika *link* pada kumpulan web dibuat kedalam graf, maka graf tersebut akan memiliki sebuah struktur menyerupai blok, karena mayoritas dari *link* tersebut bersifat *intra-host*, merujuk halaman yang masih di dalam *host* yang sama. Oleh karena itu, jika dilakukan perjalanan acak pada kumpulan web tersebut dapat dilihat sebagai rantai Markov *Nearly Completely Decomposable* (NCD) (Zhu dkk., 2005). Rantai Markov NCD adalah rantai Markov yang dapat dipartisi sehingga peluang dari keadaan awal ke keadaan selanjutnya lebih sering menunjuk keadaan yang berada di dalam partisinya dibandingkan di luar partisinya (Kontovasilis dan Mitrou, 1995). Oleh karena itu, dasar dari algoritma DPC adalah mengelompokan halaman web berdasarkan domainnya (Zhu dkk., 2005).

Masalah dari algoritma Pagerank biasa adalah besarnya memori utama yang dibutuhkan untuk bisa menyimpan matriks X (lihat persamaan 2). Oleh karena itu, dirumuskan algoritma Pagerank terdistribusi. DPC, dirumuskan oleh Zhu dkk. (2005), memakai mekanisme interaksi sederhana antara *cluster* dan lalu lintas komunikasi rendah. Ditinjau dari perspektif matematika, dibuktikan bahwa algoritma DPC setara dengan metode *Iterative Aggregation-Disaggregation* (IAD) dengan *Block Jacobi smoothing*. DPC juga memiliki keunggulan dibandingkan dengan algoritma Pagerank biasa yaitu, matriks-matriks dipecah menjadi matriks agregasi dan matriks lokal sehingga ukurannya cukup kecil untuk disimpan di memori utama, sehingga mempercepat komputasi karena tiap iterasi memerlukan sedikit operasi I/O pada *disk*. Selanjutnya, Vektor Pagerank lokal konvergen lebih cepat pada beberapa *cluster* tertentu, berbeda dengan Pagerank biasa karena terdapat komputasi tidak perlu pada *cluster* yang sudah konvergen (Zhu dkk., 2005).

Sebelum langsung membahas algoritma DPC, akan didefinisikan beberapa notasi terlebih dahulu. $e = (1, ..., 1)^T$. Misal G adalah himpunan bilangan bulat $\{1, ..., N\}$. Misal $G_1, ...G_n, n \leq N$ adalah grup agregasi dari elemen-elemen di G. Himpunan-himpunan $G_i, i = 1, ..., n$, adalah saling lepas dan $\cup_{i=1}^{n} G_i = G$. Misal N_i adalah ordo dari himpunan G_i, atau jumlah elemen-elemen di G_i (Zhu dkk., 2005).

Misal R adalah matriks agregasi $n \times N$, yang memenuhi persamaan 4 (Zhu dkk., 2005).

$$R_{ij} = \begin{cases} 1 & j \in G_i \\ 0 & \text{lainnya} \end{cases} \tag{4}$$

Dilakukan partisi pada vektor positif π menjadi $(\pi_1^T, \pi_2^T, ..., \pi_n^T)^T$ berdasarkan $\{G_i\}$. π_i adalah subvektor dengan dimensi N_i (Zhu dkk., 2005).

Maka dapat didefinisikan matriks disagregasi $S(\pi)$ $N \times n$ sebagai persamaan 5 (Zhu

dkk., 2005).

$$S(\pi) = \begin{pmatrix} S(\pi)_1 & 0 & \dots & 0 \\ 0 & S(\pi)_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & S(\pi)_n \end{pmatrix} \quad (5)$$

Dimana $S(\pi)_i$ adalah sebuah kolom vektor yang mewakili *censored stationary distribution* dari halaman-halaman di *cluster* G_i. Ingat bahwa $RS(\pi) = I$ (Zhu dkk., 2005).

Matriks transisi P dipartisi menjadi beberapa blok berdasarkan $\{G_i\}$ menjadi seperti persamaan 6 (Zhu dkk., 2005).

$$P = \begin{pmatrix} P_{11} & P_{12} & \dots & P_{1n} \\ P_{21} & P_{22} & \dots & P_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ P_{n1} & P_{n2} & \dots & P_{nn} \end{pmatrix} \quad (6)$$

Dilambangkan blok baris ke-i sebagai persamaan 7

$$P_{i*} \triangleq (P_{i1}, \dots, P_{in}) \quad (7)$$

dan dilambangkan blok kolom ke-i sebagai persamaan 8

$$P_{*i} \triangleq \begin{pmatrix} P_{1i} \\ \vdots \\ P_{ni} \end{pmatrix} \quad (8)$$

Setiap blok diagonal P_{ii} adalah matriks persegi dan merupakan matriks *intra-cluster link* dari *cluster* G_i, sementara blok-blok di luar diagonal merupakan struktur *link* antar-*cluster*. Selanjutnya, matriks agregat $A = RPS(\pi)$ adalah matriks transisi antar *cluster*. Maka dapat dibuat algoritma DPC pada algoritma 2 (Zhu dkk., 2005).

Algoritma 2 Algoritma DPC (Zhu dkk., 2005)

1: Buat matriks transisi lokal untuk tiap *cluster* G_i berdasarkan P

$$Q_i = LocalTransitionMatrix(G_i) \forall G_i \in G$$

2: $\pi_i^0 = Pagerank(Q_i, \frac{e}{N_i}, \epsilon) \forall G_i \in G$
3: Inisialisasi $k = 0$
4: $A^k = RPS(\pi^k)$
5: $z^k = Pagerank(A^k, \frac{e}{n}, \epsilon)$
6: $\forall G_i \in G$ buat sebuah *extended local transition matrix* $(N_i + 1) \times (N_i + 1)$. Dimana skalar α^k membuat jumlah nilai kolom dari B_i^k adalah satu

$$B_i^k = \begin{pmatrix} P_{ii} & \frac{(P_{i*}S(\pi^k)z^k - P_{ii}\pi_i^k z_i)}{(1-z_i^k)} \\ e^T P_{*i} & \alpha^k \end{pmatrix}$$

7: Hitung vektor *extended local Pagerank*. Dimana β_i^{k+1} adalah skalar

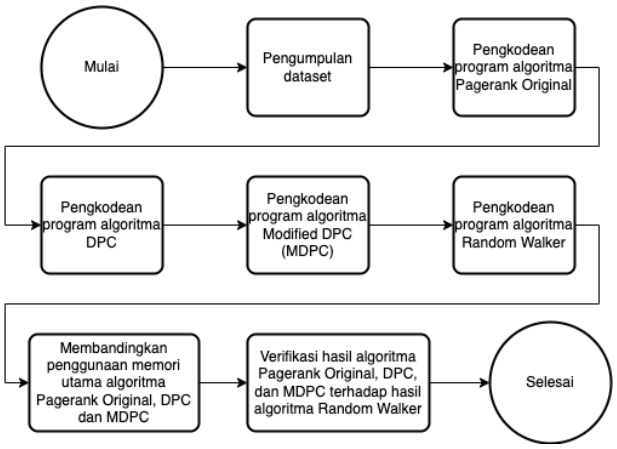
$$\begin{pmatrix} \omega_i^{k+1} \\ \beta_i^{k+1} \end{pmatrix} = Pagerank(B_i^k, \frac{e}{(N_i+1)}, \epsilon)$$

8: $\pi_i^{\sim k+1} = \frac{1-z_i^k}{\beta_i^{k+1}} \omega_i^{k+1}$
9: $\pi^{k+1} = \frac{\pi^{\sim k+1}}{||\pi^{\sim k+1}||_1}$
10: $k = k + 1$
11: Jika $||\pi^{k+1} - \pi^k|| < \epsilon$ terpenuhi, berhenti dan berikan π^k sebagai hasil akhir. Jika sebaliknya, kembali ke langkah 4

III. METODOLOGI

Tahapan Penelitian

Terdapat tahapan-tahapan yang harus dilalui untuk melaksanakan penelitian ini.



Gambar 7: *Diagram tahapan penelitian*

Tahapan penelitian dapat dilihat pada diagram 7. Terdapat beberapa algoritma yang belum dijelaskan sebelumnya seperti Modified DPC (MDPC) dan Random Walker. Algoritma MDPC dirumuskan karena kekurangan dari algoritma DPC yang akan dijelaskan pada bagian selanjutnya. Sedangkan algoritma Random Walker merupakan program yang mensimulasikan pergerakan kunjungan halaman web dan merupakan basis dari algoritma Pagerank (Page dkk., 1999), sehingga sangat cocok untuk dijadikan sebagai acuan untuk melakukan verifikasi hasil.

Pengumpulan Dataset

Pada penelitian ini digunakan dua *dataset*. *Dataset* pertama yang nantinya disebut sebagai Dataset 1 merupakan *dataset* yang diperoleh dengan cara *crawling*. Memiliki 20.493 baris halaman web *page_information* dan 2.915.842 baris *page_linking*. Data *page_information* pada Dataset 1 dapat dikelompokan ke dalam 560 *cluster* berdasarkan *domain*-nya yang dapat dilihat pada tabel 2.

Yang kedua, Dataset 2 merupakan *dataset* kecil dan *domain* kecil yang sengaja dikumpulkan untuk melihat perbedaan performa algoritma antara *dataset* yang berisi banyak *domain* besar dengan *dataset* yang berisi *domain* kecil. Batasan pada tiap *domain* yang dipakai ketika mengumpulkan data

Tabel 2: *Data cluster pada Dataset 1. Simbol "-" merupakan potongan nama domain karena terlalu panjang*

Domain	Halaman
detik.com	2.215
unj.ac.id	2.208
sport.detik.com	1.279
finance.detik.com	1.098
repository.unj.ac.id	1.089
news.detik.com	802
oto.detik.com	779
inet.detik.com	671
support.google.com	630
food.detik.com	626
⋮	⋮
-competitions.withgoogle.com	1
googledevelopers.blogspot.com	1
skillshop.exceedlms.com	1

Tabel 3: *Data cluster pada Dataset 2*

Domain	Halaman
unj.ac.id	20
ppid.unj.ac.id	20
fip.unj.ac.id	20
fbs.unj.ac.id	20
fmipa.unj.ac.id	20

adalah 20 halaman web per *domain*. Pada Dataset 2 terdapat 100 baris *page_information*, 5.944 baris *page_linking*, serta 5 *cluster* atau *domain*. Data *cluster* pada Dataset 2 dapat dilihat pada tabel 3.

Algoritma Modified DPC

Diusulkan algoritma modifikasi dari algoritma DPC atau Modified DPC (MDPC). Secara garis besar MDPC hanya memangkas langkah-langkah pada algoritma DPC, berdasarkan gagasan utama DPC yaitu melakukan perhitungan terpisah *ranking* halaman web berdasarkan *domain*-nya, dan melakukan perhitungan penggabungan dengan menghitung *ranking domain* itu sendiri. Masalah utama dari algoritma DPC

adalah langkah memperoleh matriks A pada langkah 4 yang melakukan perkalian matriks P secara utuh.

Definisi matriks A versi MDPC yang selanjutnya akan berganti notasi menjadi A_{mdpc} dapat dilihat pada persamaan 9. Makna notasi P_{**} merupakan partisi matriks transisi P yang sudah didefinisikan pada persamaan 6, sedangkan makna dari *sum* adalah nilai total dari elemen matriks.

$$A_{mdpc} = \begin{pmatrix} \frac{\text{sum}(P_{11})}{\text{sum}(P_{*1})} & \cdots & \frac{\text{sum}(P_{1i})}{\text{sum}(P_{*i})} \\ \vdots & \ddots & \vdots \\ \frac{\text{sum}(P_{i1})}{\text{sum}(P_{*1})} & \cdots & \frac{\text{sum}(P_{ii})}{\text{sum}(P_{*i})} \end{pmatrix} \quad (9)$$

Langkah-langkah algoritma MDPC dapat dilihat pada algoritma 3.

Algoritma 3 Algoritma MDPC

1: Buat matriks transisi lokal $N_i \times N_i$ untuk tiap *cluster* G_i berdasarkan P

$$Q_i = LocalTransitionMatrix(G_i) \forall G_i \in G$$

2: $\pi_i = Pagerank(Q_i, \frac{e}{N_i}, \epsilon) \forall G_i \in G$
3: $z = Pagerank(A_{mdpc}, \frac{e}{n}, \epsilon)$
4: $\pi_i \leftarrow \pi_i \times z_i \forall i \in 1 \ldots n$
5: kembalikan π sebagai hasil akhir. π adalah vektor gabungan $N \times 1$ dari semua π_i.

Algoritma Random Walker

Basis intiutif dari algoritma Pagerank adalah *random walk* pada graf. Versi penyederhanaan dalam bentuk distribusi probabilitas *random walk* pada sebuah graf web (Page dkk., 1999). Akan dibuat sebuah program yang mensimulasikan dari proses *random walk* ini yang disebut Algoritma Random Walker. Algoritma Random Walker dipakai untuk membandingkan hasil perhitungan algoritma DPC, MDPC, dan Pagerank asli. Langkah-langkah pada Algoritma Random Walker dapat dilihat pada algoritma 4.

Algoritma 4 Algoritma Random Walker

1: Tentukan jumlah iterasi dan jumlah *walker* awal-awal pada setiap halaman web
2: Instansiasi *walker* sama banyak pada setiap halaman web
3: Bentuk matriks P sesuai persamaan 3
4: Untuk setiap *walker*, pindahkan *walker* ke halaman web selanjutnya berdasarkan peluang P_{*i}, untuk i adalah indeks halaman web *walker* saat ini.
5: Ulangi langkah 4 sebanyak jumlah iterasi. *Ranking* halaman web ditentukan pada banyaknya jumlah *walker* yang tinggal.

Pada penelitian ini, algoritma Random Walker tidak dipakai untuk membandingkan performa, melainkan sebatas pembanding hasil dari algoritma Pagerank, DPC, dan MDPC.

Tahapan Pengembangan

Akan dilakukan pengkajian efisiensi penggunaan memori utama pada algoritma Pagerank, DPC, dan MDPC. Penelitian ini merupakan sub-penelitian dari penelitian utama tentang pengembangan *search engine*. Telah dilakukan penelitian sebelumnya oleh Qoriiba (2021) yang membuat modul *crawler* dan modul pendukung lain. Selanjutnya Khatulistiwa (2022) menggabungkan modul *crawler* Qoriiba (2021), Google Pagerank, dan *searcher* berbasis TF IDF menjadi *search engine* berbasis konsol. Di saat yang berdekatan, Pratama (2022) membuat modul *indexer* menggunakan *Induced Generalized Suffix Tree* dan Zalghornain (2022) menggunakan *Continuous-Bag-of-Words* dan Model *Continuous Skip-Gram*.

Karena penelitian ini merupakan sub-penelitian, maka *stack* teknologi yang digunakan dalam implementasi penelitian ini akan disamakan dengan penelitian induk

yaitu menggunakan bahasa pemrograman Python 3 dan basis data MySQL. Implementasi algoritma akan mengikuti algoritma-algoritma yang sudah dijelaskan sebelumnya. Perkiraan pengerjaan adalah satu bulan, dan tidak menutup kemungkinan akan memakan waktu lebih lama atau lebih cepat tergantung kesulitan dan kompleksitas yang dihadapi.

Untuk menguji penggunaan memori, waktu eksekusi, dan hasil algoritma *ranking* halaman web maka dilakukan langkah-langkah berikut:

1. Dilakukan pengkodean empat program algoritma yaitu Pagerank Original, DPC, MDPC, dan Random Walker
2. Ditentukan nilai masukan *damping factor*, toleransi *error*, serta dataset yang sama
3. Jalankan program Pagerank Original, DPC, MDPC, dan Random Walker terhadap dataset 1 (20.493 Halaman)
4. Bandingkan lamanya waktu, dan penggunaan memori utama algoritma Pagerank Original, DPC, dan MDPC
5. Verifikasi kemiripan hasil antara algoritma Pagerank Original, DPC, dan MDPC terhadap algoritma Random Walker
6. Jalankan program Pagerank Original, DPC, MDPC, dan Random Walker terhadap dataset 2 (100 Halaman)
7. Bandingkan lamanya waktu, dan penggunaan memori utama algoritma Pagerank Original, DPC, dan MDPC
8. Verifikasi kemiripan hasil antara algoritma Pagerank Original, DPC, dan MDPC terhadap algoritma Random Walker

Metode Perbandingan

Terdapat tiga hal yang dibandingkan pada penelitian ini, yang pertama adalah perbandingan penggunaan memori utama, lama waktunya algoritma dijalankan, dan kemiripan hasil yang dikeluarkan oleh ketiga algoritma. Tidak ada metode khusus untuk membandingkan penggunaan memori utama, dan lama waktu eksekusi. Ketiga algoritma hanya dijalankan lalu dapat dilihat dan dibandingkan penggunaan memori utama dan lamanya waktu eksekusi tiap algoritma. Sedangkan untuk kemiripan hasil, penelitian ini mengikuti metode yang dipakai oleh Zhu dkk. (2005), digunakan perhitungan jarak *Kendall's* τ atau dapat disebut sebagai *KDist*.

Rumus dari jarak *KDist* antara vektor *ranking* halaman web π terhadap $\hat{\pi}$ secara berurutan dapat dilihat pada persamaan 11, di mana $0 \leq KDist(\pi, \hat{\pi}) \leq 1$. Semakin kecil nilai jaraknya maka dianggap lebih mirip.

$$K_{ij}(\pi, \hat{\pi}) = \begin{cases} 1 & \pi_i \geq \pi_j \text{ and } \hat{\pi}_i < \hat{\pi}_j \\ 1 & \pi_i < \pi_j \text{ and } \hat{\pi}_i \geq \hat{\pi}_j \\ 0 & \text{lainnya} \end{cases} \tag{10}$$

$$KDist(\pi, \hat{\pi}) = \frac{\Sigma_{1 \leq i < j \leq N} K_{ij}(\pi, \hat{\pi})}{\frac{1}{2}N(N-1)} \tag{11}$$

IV. Hasil

Setelah program berhasil dibuat dan dijalankan terhadap semua *dataset*, maka diperoleh hasil berikut. Pada Dataset 2 puncak penggunaan memori pada ketiga algoritma relatif sama, hal ini dikarenakan data yang cukup kecil dibandingkan *overhead* memory pada program. Selanjutnya pada Dataset 1 puncak penggunaan memori terbesar ada pada algoritma pagerank asli yaitu sebesar 3,4 GB dimulai saat matriks 20.493×20.493 P dibentuk. Dengan perhitungan setiap sel matriks merupakan angka desimal *floating point* 64bit maka besarnya matriks P pada memori adalah $\pm$3,4 GB.

Selanjutnya, pada algoritma DPC penggunaan memori terbesar adalah 842 MB

Tabel 4: *Puncak penggunaan memori (dalam Mega Byte) dan waktu eksekusi (dalam detik)*

Algoritma	Puncak Memori	Waktu
Dataset 1		
PR	3.417,632	328,803
DPC	842,482	1.151,628
MDPC	86,582	816,375
Dataset 2		
PR	2,045	0,568
DPC	2,056	19,372
MDPC	2,055	0,652

yang terjadi pada pembentukan dan penyimpanan matriks P_{*i} ke *cache*. Mengingat domain terbesar di Dataset 1 yaitu "detik.com" dengan jumlah 2.215 halaman, maka dimensi matriks P_{*i} terbesar adalah 20.493×2.215 yang akan memakan memori sebesar ± 363 MB, dan ketika akan disimpan ke dalam *cache* menggunakan *pickle library* akan dilakukan penyalinan objek sebelum ditulis ke dalam *cache*, sehingga penggunaan memori $2 \times \pm 363MB = \pm 726MB$.

Pada algoritma MDPC puncak memori sebesar ± 86 MB, penggunaan memori puncak terjadi pada langkah pembentukan dan penyimpanan matriks Q ke *cache*. Karena matriks Q adalah matriks transisi lokal suatu domain, maka matriks Q terbesar adalah matriks Q domain "detik.com" dengan 2.215 halaman. Ukuran matriks Q yang terbentuk adalah 2.215×2.215 yang secara ukuran di memori adalah ± 39 MB, dan di tengah proses penyimpanan ke dalam *cache* dilakukan penyalinan objek sementara, sehingga ukuran memorinya menjadi dua kali lipat yaitu ± 86 MB. Perbandingan puncak memori dan lamanya waktu eksekusi algoritma Pagerank Original, algoritma Distributed Pagerank Computation (DPC), dan algoritma Modified DPC (MDPC), dapat dilihat pada tabel 4.

Selanjutnya akan dihitung nilai kemiripan antara nilai Pagerank yang dihasilkan program Pagerank asli, DPC, MDPC, dan Random Walker. Nilai Pagerank berbentuk vektor dan diurutkan berdasarkan *id* halaman. Pada Dataset 1, nilai *KDist* tiap vektor *ranking* halaman web yang dihasilkan oleh algoritma Pagerank, DPC, MDPC, dan Random Walker dapat dilihat pada tabel 5.

Tabel 5: *Nilai jarak Kendall vektor Pagerank pada dataset 1 (20.493 halaman). Ket: Pagerank asli (PR), Random Walker (RW).*

	PR	DPC	MDPC	RW
PR	0,0	0,250	0,260	0,027
DPC	0,250	0,0	0,277	0,252
MDPC	0,260	0,277	0,0	0,264
RW	0,027	0,252	0,264	0,0

Vektor yang paling mirip atau memiliki nilai *KDist* terkecil adalah vektor Pagerank dan Random Walker yaitu sebesar $0,02716$ atau $2,7$ persen perbedaan urutan. Sedangkan *KDist* untuk DPC terhadap Pagerank dan Random Walker secara berurutan adalah $0,24956$ dan $0,25208$ atau 25 persen dan $25,2$ persen perbedaan urutan. Selanjutnya untuk *KDist* MDPC terhadap DPC, Pagerank, dan Random Walker secara berurutan adalah $0,27681$, $0,25985$, dan $0,26387$ atau $27,7$ persen, 26 persen, dan $26,4$ persen perbedaan urutan.

Pada Dataset 2, nilai *KDist* tiap vektor *ranking* halaman web yang dihasilkan oleh algoritma Pagerank, DPC, MDPC, dan Random Walker dapat dilihat pada tabel 6. Vektor yang paling mirip atau memiliki nilai *KDist* terkecil adalah vektor Pagerank dan Random Walker yaitu sebesar $0,03293$ atau $3,3$ persen perbedaan urutan. Sedangkan *KDist* untuk DPC terhadap Pagerank dan Random Walker secara berurutan adalah $0,17131$ dan $0,19131$ atau $17,1$ persen dan $19,1$ persen perbedaan urutan. Selanjutnya untuk *KDist* MDPC terhadap DPC, Pagerank, dan Random Walker secara berurutan adalah $0,11091$, $0,12586$, dan $0,14949$ atau $11,1$ persen, $12,6$ persen, dan $14,9$ persen perbedaan urutan.

Tabel 6: *Nilai jarak Kendall vektor Pagerank pada dataset 2 (100 halaman). Ket: Pagerank asli (PR), Random Walker (RW).*

	PR	**DPC**	**MDPC**	**RW**
PR	0,0	0,171	0,126	0,033
DPC	0,171	0,0	0,111	0,191
MDPC	0,126	0,111	0,0	0,149
RW	0,033	0,191	0,149	0,0

Perbedaan hasil cukup signifikan antara Distributed Pagerank Computation (DPC) dan Modified DPC (MDPC) terhadap Pagerank Original dan Random Walker disebabkan karena dasar dari perhitungannya yang berbeda. DPC dan MDPC memisahkan halaman web berdasarkan domain-nya, sedangkan Pagerank dan Random Walker melakukan perhitungan secara utuh.

Dapat disimpulkan, algoritma Pagerank Original paling cepat dan paling mirip hasilnya dengan algoritma Random Walker dalam melakukan pe-*ranking*-an halaman web, namun memerlukan memori utama yang lebih besar. Sedangkan algoritma DPC dan MDPC memerlukan memori utama lebih kecil, sehingga cocok untuk komputer satu mesin dengan memori utama terbatas, tetapi dengan konsekuensi perbedaan hasil yang cukup besar terhadap algoritma Pagerank Original dan algoritma Random Walker dan waktu eksekusi yang lebih lambat.

V. KESIMPULAN DAN SARAN

Kesimpulan

Berdasarkan hasil implementasi dan pengujian program Algoritma Pagerank, DPC, MDPC, dan Random Walker, maka diperoleh kesimpulan sebagai berikut:

1. Algoritma Pagerank merupakan algoritma untuk menghitung *ranking* halaman web yang berbasis pada Random Walk di graf halaman web (Page dkk., 1999). Algoritma Pagerank memiliki masalah pada penggunaan memori yang besar. Algoritma DPC yang memakai metode *divide and conquer* (Zhu dkk., 2005) dipakai untuk menjawab permasalahan pada algoritma Pagerank. Algoritma MDPC merupakan modifikasi dari algoritma DPC yang dirumuskan pada penelitian ini karena terdapat langkah-langkah yang bisa disederhanakan pada algoritma DPC. Program simulasi Random Walker dibuat untuk menjadi pembanding dari hasil algoritma Pagerank, DPC, dan MDPC.
2. Dari hasil pengujian algoritma pe-*ranking*-an halaman web tercepat dalam waktu eksekusi dipegang oleh algoritma Pagerank, sedangkan dari segi penggunaan memori puncak, MDPC dan DPC jauh lebih kecil dibandingkan algoritma Pagerank. Walaupun demikian, setelah dilakukan uji hasil *ranking* dengan menghitung nilai *KDist* antara masing-masing algoritma, hasil dari algoritma Pagerank sangat mirip dengan hasil dari algoritma Random Walker dibandingan dengan algoritma DPC, dan MDPC terhadap Random Walker. Sehingga dapat disimpulkan algoritma DPC dan MDPC cocok untuk komputer satu mesin dengan memori utama terbatas, tetapi dengan mengorbankan kemiripan hasil dan waktu eksekusi lebih lambat.

Saran

Adapun saran untuk penelitian selanjutnya adalah:

1. Menguji keempat algoritma lebih lanjut dengan dataset yang lebih besar dan beragam
2. Mencari algoritma alternatif lain dari Pagerank yang secara performa memori

dan waktu lebih baik, tetapi perbedaan hasil *ranking* di bawah 10 persen

DAFTAR PUSTAKA

Amstrong, M. (2021). How many websites are there? https://bit.ly/3SbG39q. Diakses pada 30-07-2022.

Berners-Lee, T., Cailliau, R., Groff, J.-F., dan Pollermann, B. (1992). World-wide web: The information universe. *Internet Research*, 2(1):52–58.

Brin, S. dan Page, L. (1998). The anatomy of a large-scale hypertextual web search engine.

GeeksForGeeks (2022). Introduction of b-tree. https://bit.ly/4bkLvQ7. Diakses pada 28-09-2022.

GlobalStatCounter (2022). Search engine market share worldwide - july 2022. https://bit.ly/4biAfnp. Diakses pada 28-09-2022.

Khatulistiwa, L. (2022). Perancangan arsitektur search engine dengan mengintegrasikan web crawler, algoritma page ranking, dan document ranking.

Kontovasilis, K. P. dan Mitrou, N. M. (1995). Markov-modulated traffic with nearly complete decomposability characteristics and associated fluid queueing models. *Advances in Applied Probability*, 27(4):1144–1185.

MySQLDoc (2022). How mysql uses indexes. https://bit.ly/3w4zGNN. Diakses pada 30-07-2022.

Page, L., Brin, S., Motwani, R., dan Winograd, T. (1999). The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab.

Pratama, Z. (2022). Perancangan modul pengindeks pada search engine berupa induced generalized suffix tree untuk keperluan perangkingan dokumen.

Qoriiba, M. F. (2021). Perancangan crawler sebagai pendukung pada search engine.

Sample, I. (2018). What is the internet? 13 key questions answered. https://bit.ly/49bDawk. Diakses pada 30-07-2022.

Techopedia (2020). Website. https://bit.ly/3w4zyhh. Diakses pada 30-07-2022.

Wilson, R. J. (1996). *Introduction to Graph Theory*. Longman Group Ltd.

Zalghornain, M. (2022). Rancang bangun sistem pencarian teks dengan menggunakan model continuous-bag-of-words dan model continuous skip-gram pada koleksi dokumen.

Zhu, Y., Ye, S., dan Li, X. (2005). Distributed pagerank computation based on iterative aggregation-disaggregation methods. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 578–585.