PageRank Algorithm: Applying Eigenvectors to Rank Web Pages

Nadhif Al Rozin 13523076
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13523076@std.stei.itb.ac.id, [2]nadhifalrozin@gmail.com

***Abstract*—Page Rank adalah algoritma yang digunakan oleh google untuk membuat search engine mereka, page rank melakukan pengurutan laman berdasarkan tautan dan kepentingannya. Cara untuk mencari urutan kepentingan ini adalah dengan mengubahnya terlebih dahulu ke bentuk graf lalu dirubah menjadi matriks transisi, kemudian dirubah menjadi matriks stokastik, kemudian melakukan iterasi menggunakan formula google matrix untuk mendapat nilai eigen vektor dominan yang menentukan urutan laman pada database.**

***Keywords*—Algoritma, Eigen, Google, Page Rank**

I. PENDAHULUAN

Dengan semakin banyaknya laman web yang ada saat ini, yang terus bertambah secara eksponensial, muncul tantangan baru untuk menyusun informasi agar dapat lebih mudah diakses oleh pengguna. Kebutuhan akan algoritma yang efisien untuk mengurutkan kepentingan dari suatu halaman berdasarkan relevansinya menjadi hal yang sangat penting, terutama bagi mesin pencari seperti Google, Bing, dan Yahoo. Algoritma pengurutan yang cepat dan akurat akan meningkatkan kualitas hasil pencarian secara signifikan.

Google mengembangkan algoritma PageRank yang menjadi standar dalam menilai seberapa penting sebuah halaman web. PageRank akan mengurutkan halaman berdasarkan probabilitas seorang pengguna diarahkan ke laman tersebut melalui klik link tautan secara acak. Algoritma ini didasarkan pada teori graf, di mana halaman direpresentasikan sebagai simpul dan tautan antar halaman direpresentasikan sebagai sisi. Pendekatan menggunakan vektor eigen dari matriks transisi dan matriks stokastik menjadi inti dari PageRank untuk mendapatkan hasil yang cepat dan akurat.

PageRank juga dirancang untuk menangani masalah tautan acak dan halaman tanpa tautan keluar (dangling nodes) menggunakan faktor damping dan pemodelan Random Surfer. Dengan cara ini, algoritma dapat memastikan bahwa hasil pencarian tetap relevan meski terdapat struktur web yang kompleks.

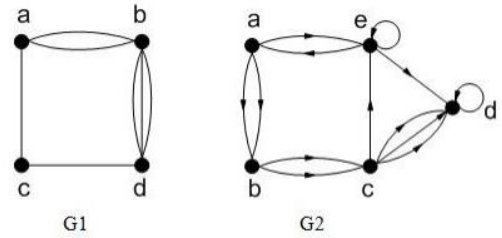
Makalah ini akan membahas bagaimana vektor eigen diterapkan dalam algoritma untuk mengurutkan halaman web secara efisien. Algoritma ini dirancang untuk menghasilkan hasil yang akurat tanpa membebani daya komputasi secara berlebihan. Dengan pendekatan ini, mesin pencari dapat memberikan pengalaman yang lebih baik kepada pengguna dalam menemukan informasi di internet. Pendekatan yang tepat dalam memanfaatkan teori graf dan algoritma PageRank memungkinkan pengelolaan data dalam jumlah besar secara efektif, sekaligus mempertahankan akurasi tinggi dalam mengurutkan halaman.

II. LANDASAN TEORI

A. Teori Graf

Graf terdiri atas 2 komponen utama, simpul (nodes) dan sisi (edges). Simpul adalah titik yang merepresentasikan suatu objek dan sisi adalah garis yang menghubungkan simpul ke simpul, sisi bisa diartikan untuk menyatakan hubungan antar simpul. Pada konteks ini simpul adalah laman dan sisi adalah tautan.

Graf berarah adalah Graf yang sisi sisinya memiliki orientasi arah. Ditunjukkan dengan tanda panah pada gambar dibawah [1].



Gambar 1 Contoh gambar graf berarah dan tak-berarah
Sumber : ppt pak Rinaldi Munir

Pada Konteks ini orientasi arah digunakan untuk menujunkan hubungan tautan keluar dan kedalam dari suatu laman.

Matriks Adjacency adalah representasi matriks dari suatu graf di mana elemen-elemen yang ada pada matriks menunjukan ada atau tidaknya (0/1) hubungan (sisi) antara dua buah simpul. Dalam PageRank, ini digunakan untuk merepresentasikan hubungan antara 1 laman dengan laman lainnya. Pada algoritma ini matriks Adjacency akan dirubah menjadi matriks transisi (Matriks stokastik)

Graf yang ada pada web umumnya memiliki simpul yang tidak memiliki tautan keluar (dead-end). Hal ini

dapat menggangu perhitungan matriks transisi karena menghasilkan kolom nol yang tidak sesuai dengan sifat dari matrik stokastik. Selain itu subgraf yang terhubung secara kuat (semua simpulnya dapat saling menjangkau) perlu di identifikasi untuk mencegah hasil yang tidak stabil. Contohnya jika ada laman yang tidak terhubung artinya nilai PageRank bisa menjadi tidak relevan untuk halaman tersebut. Dengan mengidentifikasi ini algoritma bisa menambahkan faktor peredaman (faktor damping), sehingga probabilitasnya menjadi 0-0 untuk dijangkau.

B. Aljabar Linear

Nilai Eigen (eigenvalue) adalah scalar λ yang memenuhi:

$$Av = \lambda v$$

Dimana A adalah matriks persegi, v adalah vektor eigen yang bukan nol dan λ adalah nilai eigen. Nilai eigen menunjukkan faktor skala yang didapat jika vektor eigen v dioperasikan dengan matriks A. jika nilai eigen lebih dari 1 artinya vektor diperbesar, jika kurang dari 1 maka diperkecil. Sedangkan jika sama dengan 1 artinya vektor tidak berubah, dalam konteks ini artinya relevan. Karena algoritma PageRank bertujuan untuk menemukan distribusi probabilitas yang stabil (tidak berubah) setelah iterasi yang cukup.

Untuk menemukan nilai eigen λ, digunakan persamaan Karakteristik:

$$\det(A - \lambda I) = 0$$

Dimana I adalah matriks identitas. Dan dicari eigen value (λ) yang memenuhi persamaan di atas. Setelah didapat nilai tersebut, dicari eigen vektor untuk tiap nilai eigen yang ditemukan [2].

Dalam Algoritma page rank, vektor eigen yang terkait dengan λ = 1 merepresentasikan distribusi probabilitas yang dalam kondisi stabil, Artinya vektor ini mewakili bobot atau peringkat setiap halaman dalam graf yang tidak akan berubah lagi. Hal ini disebut vektor PageRank.

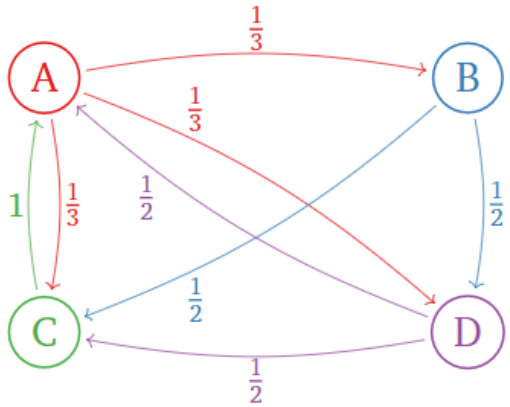
Vektor Page Rank adalah vektor yang digunakan untuk menentukan peringkat halaman web, awalnya vektor akan diberi nilai awal yang sama, nilai itu akan diperbarui berdasarkan jumlah dan kualitas dari tautan yang masuk ke laman, iterasi ini akan dilakukan hingga nilai page rank mencapai kondisi stabil (konvergen)

Matriks Stokastik adalah konsep dalam matematika yang digunakan untuk memodelkan sistem probabilistik, secara umum matriks ini digunakan untuk menggambarkan kemungkinan adanya transisi dari suatu keadaan ke keadaan yang lain dalam suatu sistem dengan basis probabilitas (kemungkinan).

Matriks ini adalah matriks yang memiliki 2 syarat utama yaitu elemennya berkisar antara 0 dan 1 dan jumlah elemen pada setiap kolom adalah 1. Dalam PageRank, matriks ini digunakan untuk memodelkan transisi antar halaman berdasarkan kemungkinan untuk berpindah dari satu halaman ke halaman lain menggunakan tautan. Pada PageRank matriks stokastik yang digunakan disebut Google Matrix, Matrix ini dibangun dengan cara menghitung probabilitas berpindah dari suatu halaman ke halaman yang lainnya berdasarkan tautan yang ada di dalam web. Kolom Matriks mewakili halaman sumber, Baris Matriks mewakili halaman tujuan dan Elemen menunjukkan kemungkinan berpindah dari halaman sumber ke tujuan.

$$A = \begin{pmatrix} .3 & .4 & .5 \\ .3 & .4 & .3 \\ .4 & .2 & .2 \end{pmatrix}$$

Gambar 2 Contoh Stochastic Matrix
Sumber: https://textbooks.math.gatech.edu/ila/stochastic-matrices.html.



Gambar 3 Representasi 4 Laman web dalam graf
Sumber: https://textbooks.math.gatech.edu/ila/stochastic-matrices.html.

Dapat dilihat jika laman A mempunyai 3 tautan, sehingga tiap tautan diberi 1/3 Rank nya diberikan ke B, C dan D. laman B mempunyai 2 tautan, sehingga tiap tautan ½ Rank nya diberikan ke C dan D. laman C mempunyai 1 tautan sehinggal Ranknya diberikan ke A seluruhnya. Laman D mempunyai 2 tautan, sehingga tiap tautan ½ Ranknya diberikan ke A dan C.

$$\begin{pmatrix} 0 & 0 & 1 & \frac{1}{2} \\ \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{2} & 0 & 0 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} c + \frac{1}{2}d \\ \frac{1}{3}a \\ \frac{1}{3}a + \frac{1}{2}b \\ \frac{1}{3}a + \frac{1}{2}b \end{pmatrix} = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}.$$

Gambar 4 Penghitungan nilai eigen di tiap iterasi
Sumber: https://textbooks.math.gatech.edu/ila/stochastic-matrices.html.

Dapat dilihat jika representasi nilai rank yang diberikan dilakukan per kolom dengan warna merah adalah A, biru B, hijau C dan ungu D. di dot kan dengan vektor v = (a, b, c, d) yang merupakan vektor eigen yang mengandung urutan kepentingan dari Laman laman di atas, yang perlu diperhatikan adalah Rank Vektor adalah vektor eigen

yang punya eigen value 1. Hasil perkalian nya adalah eigen vektor, jika nilai eigen adalah 1 maka vektor tersebut adalah Eigen vektor yang dominan.

Eigen vektor yang dominan pada algoritma Page Rank adalah kondisi Dimana eigen vektor memiliki nilai eigen 1, yang artinya ada dalam kondisi stabil, kondisi setimbang di sistem.

Perron-Frobenius theorem adalah teorema yang berlaku di matriks Stokastik yang memberikan properti penting dalam algoritma ini. eigen value terbesar pada matriks ini adalah 1, yang artinya eigen vektor yang dominannya adalah eigen vektor dengan eigen value 1, dan ini menggambarkan distribusi probabilitas yang stabil (steady-state distribution). Konvergensi ini hanya bisa dijamin jika semua elemen matriks bisa dijangkau dan aperiodic (tidak ada pola siklik di graf) hal ini agar tidak berputar disuatu lingkaran dan tidak menemukan dead-end.

C. Probabilistik

Random Surfer adalah pendekatan probabilistic yang digunakan dalam algoritma Page Rank untuk memodelkan perilaku seorang pengguna internet yang secara acak menjelajahi laman laman di web. Model ini digunakan untuk menjelaskan probabilitas distribusi page dihitung berdasarkan pola jelajah pengguna.

Perilaku pengguna (surfer) di modelkan dengan 2 aturan. Pada awalnya pengguna memulai dari sebuah halaman web yang acak, dan memiliki 2 pilihan untuk menekan suatu tautan terkait atau melompat ke halaman acak lain.

Faktor Damping adalah probabilitas pengguna untuk mengikuti tautan yang ada di halaman terkait, pada umumnya diatur menjadi 0.85 [3], yang dapat diartikan dari 100 orang pengguna, 85 orang akan mengikuti link dan 15 orang lainnya akan ke laman acak lain. Oleh karena itu peluang pengguna melompat ke acak dimodelkan dengan 1 – d

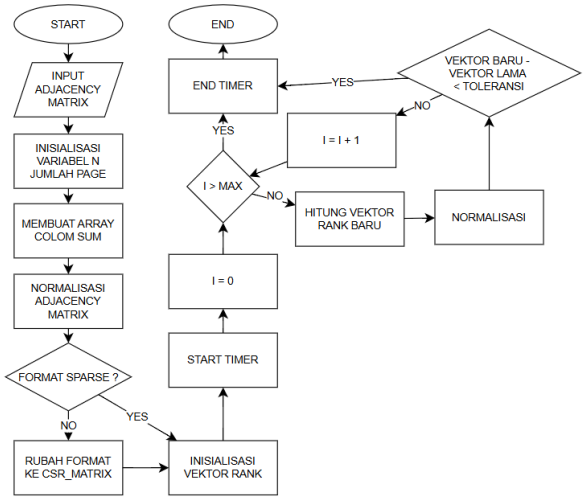
Matriks Google adalah model Random Surfer yang merupakan kombinasi dari Matriks transisi yang merepresentasikan probabilitas pengguna mengikuti link dan Matriks lompatan acak, Dimana setiap laman punya probabilitas yang sama untuk dikunjungi secara acak.

$$G = d \cdot M + (1 - d) \cdot \frac{1}{n} \cdot \mathbf{1}\mathbf{1}^T$$

Ini adalah formula google Matrix yang menggabungkan Matrix transisi (M) dengan faktor damping (d) ditambahkan dengan faktor melompat acak (1 – d) dengan *uniform jump probability* n adalah jumlah laman yang ada. 10000 adalah skala kecil, 10000 – 1000000 adalah skala medium dan lebih dari itu adalah skala besar

III. IMPLEMENTASI

Implementasi Algoritma PageRank dapat dilihat sebagai berikut:



Gambar 5 Flowchart Implementasi

Program dimulai dengan menginput adjacency matrix, pada implementasi ini digunakan random untuk mengisi 10000 x 10000 matrix dengan nilai 0 / 1. Setelah itu inisialisasi semua variabel dan normalisasi matrix berdasarkan jumlah elemen per kolom menjadi 1. Lalu dipastikan jika matrix sudah dalam format csr_matrix. Kemudian dimulai iterasi, di tiap iterasi akan dibuat vektor rank yang baru dengan mengikuti formula Matriks Google, setelah dinormalisasi, jika selisihnya dengan vektor lama di atas toleransi (ketelitian) maka iterasi dimulai lagi dengan vektor baru menggantikan variabel vektor rank. Yang dilakukan dari Implementasi ini adalah mendekati hasil vektor rank tanpa menghitung satu per satu eigen vektor nya, namun mencari nilai eigen yang mendekati 1, artinya tidak akan berubah lagi meskipun di loop lagi.

```python
def page_rank(adjacency_matrix, damping_factor=0.85, max_iter=100, tol=1e-6):
    """
    n = adjacency_matrix.shape[0]

    column_sums = np.array(adjacency_matrix.sum(axis=0)).flatten()
    column_sums[column_sums == 0] = 1  # Avoid division by zero

    # Create a diagonal matrix with the reciprocal of column sums
    diagonal_scaling = csr_matrix((1 / column_sums, (np.arange(len(column_sums)), np.arange(len(column_sums)))

    # Normalize adjacency matrix columns
    transition_matrix = adjacency_matrix @ diagonal_scaling  # Efficient sparse matrix multiplication

    # Convert to a sparse format if not already sparse
    if not isinstance(transition_matrix, csr_matrix):
        transition_matrix = csr_matrix(transition_matrix)

    # Initialize rank vector
    rank_vector = np.ones(n) / n

    # Start timer
    start_time = time.time()

    # iteration
    for iteration in range(max_iter):
        new_rank_vector = damping_factor * transition_matrix.dot(rank_vector) + (1 - damping_factor) / n
        if np.linalg.norm(new_rank_vector - rank_vector, ord=1) < tol:
            print(f"Converged after {iteration + 1} iterations.")
            break
        rank_vector = new_rank_vector

    # End timer
    end_time = time.time()
    computation_time = end_time - start_time

    return rank_vector, computation_time
```

Gambar 6 Source Code Implementasi Page Rank

```python
def page_rank_without_iteration(adjacency_matrix):
    """
    Computes the PageRank using the eigenvector method.
    :param adjacency_matrix: Sparse adjacency matrix (numpy array or scipy.sp
    :return: PageRank vector and computation time.
    """

    column_sums = np.array(adjacency_matrix.sum(axis=0)).flatten()
    column_sums[column_sums == 0] = 1  # Avoid division by zero

    diagonal_scaling = csr_matrix((1 / column_sums, (np.arange(len(column_sum

    transition_matrix = adjacency_matrix @ diagonal_scaling

    if not isinstance(transition_matrix, csr_matrix):
        transition_matrix = csr_matrix(transition_matrix)

    # Start timer
    start_time = time.time()

    # Compute PageRank
    eigenvalues, eigenvectors = np.linalg.eig(transition_matrix.toarray())
    eigenvalue_index = np.argmax(eigenvalues)
    rank_vector = np.abs(eigenvectors[:, eigenvalue_index])
    rank_vector /= rank_vector.sum()  # Normalize

    # End timer
    end_time = time.time()
    computation_time = end_time - start_time

    return rank_vector, computation_time
```

Gambar 7 Source Code Implementasi Page Rank Tanpa Iterasi

Untuk tanpa iterasi, metode yang dilakukan adalah mencari semua eigen vektor dan eigen value yang ada dari matriks transisi dan mencari eigen vektor dari nilai eigen yang paling besar. Eigen vektor tersebut adalah vektor rank. Dilakukan hal yang sama dengan time untuk mencatat waktu yang digunakan untuk melakukan komputasi

```python
def make_adjacency_matrix(n_pages, density):
    """
    Creates a random adjacency matrix.
    :param n: Number of nodes.
    :param p: Probability of an edge between two nodes.
    :return: Sparse adjacency matrix (numpy array or scipy.sparse.csr_matrix).
    """

    # Random untuk testing
    adjacency_matrix = random(n_pages, n_pages, density=density, format='csr', random_state=42, data_rvs=np.ones)
    np.fill_diagonal(adjacency_matrix.toarray(), 0)  # Remove self-links
    # Konversi elemen menjadi bilangan bulat (0 atau 1)
    adjacency_matrix.data = (adjacency_matrix.data > 0).astype(int)

    # End timer
    end_time = time.time()
    print("Adjacency matrix created.")
    print(f"Matrix size: {n_pages}x{n_pages}")
    print(f"Density: {density}")
    print(f"Matrix creation time: {end_time - start_time:.6f} seconds.")
    return adjacency_matrix
```

Gambar 8 Source Code Implementasi Random Matrix

Perlu diketahui, radom state yang digunakan untuk testing adalah 42, untuk konsistensi mengetes 2 jenis implementasi page rank.

Lebih jelasnya dapat dilihat di github berikut : https://github.com/Narrr21/Page-Rank-Algeo/blob/main/README.md.

IV. HASIL DAN PEMBAHASAN

Pengetesan dilakukan dengan kode yang ada di Implementasi dan random state yang digunakan adalah 42, akan dilakukan pengujian dengan 2 parameter dan 2 nilai untuk perbandingan hasil. Parameter yang digunakan adalah besar dataset berupa 4 laman, 100 laman, dan 10000 laman, selain parameter itu metode page rank yang akan digunakan juga menjadi penting, metode yang digunakan ada 2 yaitu tanpa iterasi dan dengan iterasi. Nilai yang akan dilihat adalah keakuratan untuk 10 page dengan rank tertinggi dan waktu yang dibutuhkan.

```
Starting make adjacency matrix...
Adjacency matrix:
[[0. 0. 1. 1.]
 [1. 0. 0. 0.]
 [1. 1. 0. 1.]
 [1. 1. 0. 0.]]

Starting PageRank computation...
Converged after 19 iterations.
PageRank computation completed.
Computation time: 0.000848532 seconds.
Top 10 PageRanks: [0 2 3 1]
```

Gambar 9 Hasil dari 4 laman menggunakan iterasi.

```
Starting make adjacency matrix...
Adjacency matrix:
[[0. 0. 1. 1.]
 [1. 0. 0. 0.]
 [1. 1. 0. 1.]
 [1. 1. 0. 0.]]

Starting PageRank computation...
PageRank computation completed.
Computation time: 0.009088516 seconds.
Top 10 PageRanks: [0 2 3 1]
```

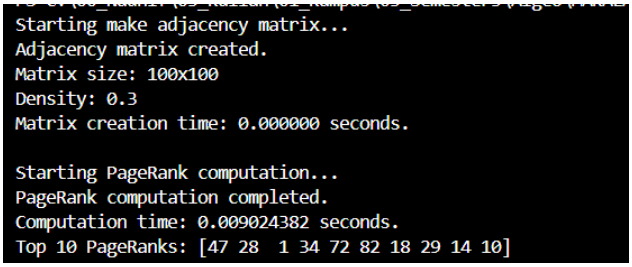
Gambar 10 Hasil dari 4 laman tanpa iterasi

Dapat direpresentasikan menjadi graf di Gambar 3.

Hasil di atas menunjukkan hasil untuk pendekatan tanpa iterasi maupun dengan iterasi, menggunakan vektor eigen dapat menghasilkan nilai yang sama. Namun jika demikian muncul pertanyaan untuk apa sebenernya dibuat dengan iterasi ? jika nilai yang dihasilkan hanya didekati dan ada kemungkinan kesalahan, terutama untuk data yang sangat besar. Hal ini dikarenakan daya komputasi yang dibutuhkan untuk menghitung eigen vektor bertumbuh secara signifikan, namun tidak dengan iterasi. Tentunya iterasi bisa menghasilkan hasil yang tidak akurat, namun Tingkat akurasi yang didapat sudah tinggi dan dengan daya komputasi yang jauh lebih kecil. Untuk akurasi sendiri bisa ditingkatkan dengan dikombinasikan dengan Teknik lain untuk meningkatkan konvergensi. Salah satunya dengan metode Fourier Domain Scoring [5]. Atau dengan memperkecil toleransi kesalahan dan meningkatkan jumlah iterasi maksimal

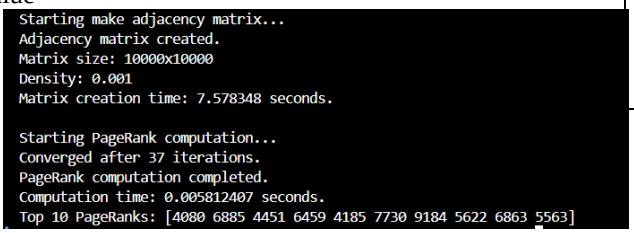
```
Starting make adjacency matrix...
Adjacency matrix created.
Matrix size: 100x100
Density: 0.3
Matrix creation time: 0.002106 seconds.

Starting PageRank computation...
Converged after 7 iterations.
PageRank computation completed.
Computation time: 0.001083136 seconds.
Top 10 PageRanks: [47 28  1 34 72 82 18 29 14 10]
```

Gambar 11 Hasil dari Matrix 100 x 100 dengan Density 0,3 dengan iterasi

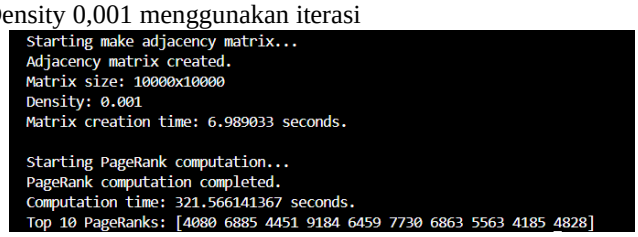


Gambar 12 Hasil dari Matrix 100 x 100 dengan Density 0,3 tanpa iterasi

Matrix 100 x 100 adalah ukuran yang relatif kecil jika dibandingkan dengan ukuran jaringan world wide web, untuk ukuran ini dapat dengan mudah dihitung langsung tanpa menggunakan pendekatan iterasi. Akan tetapi, apabila jumlah halaman bertumbuh secara signifikan, waktu yang digunakan akan bertumbuh secara eksponensial. Hal ini disebabkan oleh kompleksitas komputasi dari pencarian tiap eigen vektor dan eigen value



Gambar 13 Hasil dari Matrix 10000 x 10000 dengan Density 0,001 menggunakan iterasi



Gambar 14 Hasil dari Matrix 10000 x 10000 dengan Density 0,001 tanpa iterasi

Mengingat jika 10.000 halaman saja masih termasuk dalam kategori skala kecil, dan waktu nya bertumbuh secara signifikan. Oleh karena itu, metode iterasi atau pendeketan menggunakan google matriks lebih efisien. Pendekatan ini tidak hanya memungkinkan penghematan waktu, tetapi juga sumber daya komputasi. Optimalisasi algoritma dan pengurangan redundansi menjadi faktor kunci untuk mengelola data yang sangat besar seperti world wide web.

Tabel 1 Tabel waktu komputasi implementasi

	4 laman	100 laman	10000 laman
Iterasi	0.00084	0.00108	0.00581
Tanpa Iterasi	0.00908	0.00902	321.56

Dapat dilihat jika penggunaan metode iterasi adalah cara yang efisien untuk mencari page dengan rank tertinggi, Dan hal ini bisa dilakukan karena page rank awal mulanya menggunakan pendekatan vektor eigen, karena jika tanpa iterasi, program harus menghitung eigen dengan nilai paling tinggi (Eigen vektor yang dominan), hal inilah yang dimanfaatkan oleh program bukan dengan mencari nilai eigen value untuk tiap eigen vektor namun dengan mendekati hasilnya, karena pada algoritma Page Rank, eigen vektor yang dominan mempunyai eigen value 1, hal ini terjadi karena sifat khusus dari matriks transisi yang digunakan untuk merepresentasikan hubungan antar halaman web. Hal ini utamanya karena matriks transisi dalam pagerank adalah matriks stokastik yang artinya jumlah elemen setiap kolom adalah 1

Tabel 2 Tabel 10 laman teratas implementasi

	4 laman	100 laman	10000 laman
Iterasi	[0, 2, 3, 1]	[47, 28, 1, 34, 72, 82, 18, 29, 14, 10]	[4080, 6885, 4451, 6459, 4185, 7730, 9184, 5622, 6863, 5563]
Tanpa Iterasi	[0, 2, 3, 1]	[47, 28, 1, 34, 72, 82, 18, 29, 14, 10]	[4080, 6885, 4451, 9184, 6459, 7730, 6863, 5563, 4185, 4828]

Dapat dilihat jika untuk 4 laman, hasil laman teratas sudah sesuai jika di cross check manual dengan menggunakan graf, hasil dari 100 laman menggunakan iterasi dan tanpa iterasi juga sama, artinya metode iterasi masih akurat, namun saat 10000 laman, ada sedikit perbedaan di urutan, namun 10 laman teratas masih beranggotakan laman yang sama, 3 laman teratas sama namun dari laman ke 4 dan seterusnya berbeda.

Hal yang paling penting dari algoritma ini adalah penggunaan matriks stokastik yang eigenvektor dominannya merepresentasikan kondisi distribusi yang stabil. Sesuai dengan persamaan berikut:

$$Av = \lambda v$$

Sesuai dengan teorema Perron-Frobenius, $\lambda = 1$ maka $Av = v$. yang menjadi dasar untuk melakukan iterasi.

Dengan demikian, penggunaan sifat eigen vektor pada Algoritma Page Rank membuat tugas search engine dengan daya komputasi yang sangat berat terutama dalam world wide web menjadi jauh lebih ringan. Algoritma inilah salah satu faktor yang membuat Google seperti saat ini.

V. KESIMPULAN

Dari hasil, dapat diperhatikan jika pendekatan dengan menggunakan iterasi untuk mencari page rank meningkatkan efisiensi komputasi, pendekatan ini memungkinkan penghitungan pada dataset yang jauh lebih besar, tentunya ada yang harus dikorbankan dari metode ini, akurasi yang didapat tidak seakurat penghitungan vektor eigen satu per satu. Namun karena hasilnya cukup akuran dan memenuhi kebutuhan praktis, metode ini lebih baik digunakan terutama dalam World Wide Web. Akurasi juga bisa ditingkatkan lebih lanjut dengan penggabungan metode lain dan penyesuaian toleransi kesalahan.

Penggunaan metode iterasi yang menggunakan

teorema Perron-Frobenius dan memanfaatkan eigen vektor dengan nilai eigen value maksimal 1 dari google matrix. Dengan menggunakan pendekatan ini beban komputasi untuk mencari eigen vektor dominan sangatlah rendah, membuat vektor eigen dan matriks stokastik menjadi kunci untuk algoritma page rank.

VI. Ucapan Terimakasih

Pertama-tama, penulis memanjatkan puji dan Syukur kepada Tuhan YME, atas berkat dan nikmatnya diberikan kelancaran untuk menyelesaikan makalah ini. Penulis juga mengucapkan terima kasih kepada kedua orang tua, serta teman teman yang sudah mendukung selama pembuatan makalah ini.

Kemudian penulis juga memberikan rasa terimakasih khusus kepada Bapak Rinaldi Munir selaku dosen pengajar mata kuliah IF2110 Aljabar dan Geometri kelas K2, yang telah memberikan penulis ilmu dan kesempatan untuk mengerjakan makalah ini.

Referensi

[1] Munir, Rinaldi. 2024. Graf (Bagian 1), https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf, diakses pada 28 Desember 2024.

[2] Munir, Rinaldi. 2024. Nilai Eigen dan Vektor Eigen (Bagian 1), https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-19-Nilai-Eigen-dan-Vektor-Eigen-Bagian1-2023.pdf, diakses pada 28 Desember 2024.

[3] L. Page, S. Brin, R. Motwani, and T. Winograd, "The Anatomy of a Large-Scale Hypertextual Web Search Engine," in *Proceedings of the Seventh International World Wide Web Conference (WWW7)*, Brisbane, Australia, 1998, pp. 107-117.

[4] Margalit, Dan, Rabinoff, Joseph. *Stochastic Matrices*, Georgia Institute of Technology, 2024. https://textbooks.math.gatech.edu/ila/stochastic-matrices.html. diakses pada 29 Desember 2024.

[5] D. Purwitasari, "A Study on Ranking Method in Retrieving Web Pages Based on Content and Link Analysis: Combination of Fourier Domain Scoring and PageRank Scoring," *Jurnal Ilmiah Teknologi Informasi*, vol. 7, no. 1, pp. 11-20, 2008. Available: neliti.com.

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 28 Desember 2024

Nadhif Al Rozin 13523076