JURNAL SISFOTENIKA
Vol. 14 No. 1, Januari 2024
p-ISSN: 2087-7897 ; e-ISSN : 2460-5344
DOI : 10.30700/sisfotenika.v14i1.410

Pengoptimalan Algoritma Genetika dengan Multithreading untuk Pencarian Kalimat

Optimizing Genetic Algorithms with Multithreading for Sentence Retrieval

Kartika Sari *[1], Muharam Rizqi Ananda[2], Muhammad Syamsul Bahri [3], Siti Oktavia Wulandari [4], Herry Prayuda [5]
[1,2,3,4,5] Jurusan Rekayasa Sistem Komputer, Fakultas MIPA Universitas Tanjungpura
Jalan Prof. Dr. H. Hadari Nawawi Pontianak
Telp./Fax : (0561) 577963
e-mail: * [1] kartika.sari@siskom.untan.ac.id,[2] muharamra@student.untan.ac.id,
[3] h1051201091@student.untan.ac.id-, [4] h1051211100@student.untan.ac.id, [5] h1051211057@student.untan.ac.id

Abstrak

Algoritma genetika sering digunakan untuk memecahkan masalah pencarian teks. Namun, kinerja waktu eksekusi algoritma genetika konvensional masih kurang efisien untuk pencarian teks berskala besar. Hal ini disebabkan proses evaluasi fitness yang dilakukan secara berulang pada setiap generasi. Penelitian ini berfokus pada peningkatan efisiensi waktu eksekusi algoritma genetika untuk pencarian teks dengan mengimplementasikan multithreading. Multithreading memungkinkan proses evaluasi fitness didistribusikan ke beberapa thread secara paralel. Pada penelitian ini dilakukan pengembangan sistem pencarian teks berbasis algoritma genetika dengan membandingkan kinerja waktu menggunakan Multithread dan Singlethread, implementasi multithreading digunakan untuk mendistribusikan evaluasi fitness, dan pengukuran kinerja waktu eksekusi. Pengujian sistem dilakukan dengan cara melakukan pencarian keyword dengan menggunakan 30 buah data uji dalam 3 (tiga) skenario pengujian, yaitu dengan menggunakan Single-thread, Multithreading 2 Core, dan Multithreading 4 core untuk kemudian akan dihitung dan dibandingkan waktu eksekusinya. Hasilnya menunjukkan perbedaan kinerja waktu antara algoritma genetika asli dengan singlethread dan yang dioptimalkan menggunakan Multithreading. Hasil pengujian didapatkan nilai waktu eksekusi sistem dengan Singlethread membutuhkan waktu rata-rata 21,66 detik per generasi. Sedangkan dengan menggunakan Multithread 2 core waktu eksekusi menjadi 21,24 detik, dan 4 core menjadi 11,48 detik. Implementasi multithreading meningkatkan kecepatan eksekusi hingga 50% karena proses fitness dievaluasi secara paralel. Algoritma dengan 4 core memberikan peningkatan kinerja waktu yang signifikan.

Kata kunci: Algoritma Genetika, *Multithreading*, Pencarian Kalimat, Efisiensi Waktu Eksekusi

Abstract

Genetic algorithms are widely used to solve text search problems. However, the execution time performance of conventional genetic algorithms is still less efficient for large-scale text searches. This is due to the fitness evaluation process being carried out repeatedly at each generation. This research focuses on increasing the efficiency of genetic algorithm execution time for text search by implementing multithreading. Multithreading allows the fitness evaluation process to be distributed across multiple threads in parallel. In this research, a genetic algorithm-based text search system was developed by comparing time performance using Multithread and Singlethread, multithreading implementation was used to distribute fitness evaluations and

measure execution time performance. System testing is carried out by searching for keywords using 30 pieces of test data in 3 test scenarios, Single-thread, Multithreading 2 Cores, and Multithreading 4 cores, and then the execution time will be calculated and compared. The results show the difference in time performance between the original genetic algorithm with a single-thread and the one optimized using Multithreading. The test results showed that the system execution time with Singlethread took an average of 21.66 seconds per generation. Meanwhile, by using Multithread 2 cores the execution time is 21.24 seconds, and 4 cores is 11.48 seconds. Multithreading implementation increases execution speed by up to 50% because fitness processes are evaluated in parallel. Algorithms with 4 cores provide significant time performance improvements.

Keywords: *Genetic Algorithm, Multithreading, Sentence Search, Execution Time Efficiency*

1. PENDAHULUAN

Di bidang ilmu komputer, algoritma genetika adalah teknik pencarian untuk menemukan perkiraan penyelesaian untuk kasus pencarian dan optimasi[1]. Algoritma ini terinspirasi oleh biologi *evolusioner*, seperti mutasi, warisan, seleksi alam, dan rekombinasi *crossover*. [2]. Salah satu aplikasi algoritma genetika adalah dalam pencarian teks atau kalimat[3]. Namun kinerja waktu algoritma genetika umumnya masih kurang efisien untuk masalah pencarian teks yang besar dengan banyak data, seperti dalam aplikasi pencarian informasi. Hal ini disebabkan oleh pemrosesan yang dilakukan secara berulang pada setiap generasi[4].

Salah satu tantangan utama yang timbul dalam penerapan algoritma genetika untuk pencarian kalimat adalah kinerja waktu eksekusi yang lambat seiring dengan kompleksitas masalah yang dihadapi. Hal ini terjadi karena operasi-operasi genetika seperti seleksi, *crossover*, dan mutasi harus dilakukan secara berulang pada populasi calon solusi yang besar untuk mencapai konvergensi[4]. Populasi yang besar diperlukan untuk memastikan keberagaman genetik dan mencegah konvergensi prematur. Namun hal ini berarti proses evaluasi *fitness* harus dilakukan berulang kali pada setiap generasi, sehingga waktu eksekusi menjadi lama.

Masalah lain yang dihadapi terkait dengan algoritma genetika adalah pada aplikasi yang melibatkan dataset besar atau evaluasi solusi yang kompleks, menciptakan hambatan dalam mendapatkan solusi optimal secara efisien. Keterbatasan waktu menjadi semakin signifikan dalam konteks aplikasi real-time atau lingkungan di mana respons cepat diperlukan. Tingkat konvergensi yang lambat dan jumlah iterasi yang banyak untuk mencapai solusi yang baik juga dapat memberikan dampak negatif pada waktu eksekusi total algoritma genetika.Masalah waktu eksekusi yang cukup lama pada algoritma genetika dapat menjadi hambatan dalam pemrosesan data besar, oleh karena itu, diperlukan pengoptimalan sistem, salah satunya dengan cara mengimplementasikan multi-threading untuk menjalankan berbagai tahap algoritma secara simultan, mempercepat proses evolusi dan meningkatkan efisiensi komputasinya.

Multithreading merupakan suatu teknik pemrograman yang memungkinkan beberapa subproses dalam program untuk berjalan secara bersamaan (paralel) [5]. Untuk menyelesaikan masalah kinerja waktu eksekusi yang lambat pada algoritma genetika untuk pencarian kalimat, maka akan dikembangkan sistem pengoptimalan dengan mengimplementasikan konsep *multithreading*. Sistem ini menerapkan algoritma genetika standar untuk mewakili populasi dan melakukan operasi-operasi genetikanya. Perbedaannya proses evaluasi *fitness* pada setiap individu didistribusikan ke beberapa *thread* secara paralel. Setiap *thread* bertanggung jawab mengevaluasi individu tertentu. Dengan demikian, proses evaluasi *fitness* yang merupakan tahap paling lama dapat dieksekusi secara paralel, sehingga diharapkan dapat mempercepat proses evolusi dan meningkatkan efisiensi waktu eksekusi keseluruhan untuk mencari kalimat yang diberikan. Kemudian melakukan perbandingan kinerja waktu antara algoritma genetika asli dan algoritma genetika yang dioptimalkan dengan *multithreading* dalam pencarian. Sehingga perbedaan kinerja waktu antara algoritma genetika asli dan algoritma genetika yang dioptimalkan

dengan *multithreading*. Setelah itu mengukur waktu eksekusi algoritma yang masing-masing dievaluasi tersendiri secara paralel.

Penelitian terkait sistem paralel pernah dilakukan oleh beberapa peneliti sebelumnya, misalnya pada penelitian yang berjudul "Model Paralelisasi Algoritma Genetika Terpandu pada Sistem Penjadwalan Kuliah Universitas dengan Alokasi Waktu Dinamis"[4]. Penelitian ini mengimplementasikan algoritma genetika dengan *multithreading* untuk menjadwalkan perkuliahan. Hasilnya menunjukkan penurunan waktu eksekusi hingga 62% dibandingkan tanpa *multithreading*. Penelitian berikutnya berjudul "Penerapan Algoritma Genetika Pada Pengenalan Paragraf[3] yang menerapkan algoritma genetika dengan menggunakan parameter dan memilih operator genetika yang tepat untuk mengenali paragraf berdasarkan target penelitian, serta mengenali naskah. Dari penelitian tersebut, didapatkan nilai jumlah individu, batas generasi, probabilitas, dan tingkat mutasi memengaruhi pengenalan paragraf.

Penerapan algoritma genetika dalam menyelesaikan tantangan pencarian teks merupakan hal yang umum. Namun, algoritma genetika konvensional mengalami kendala yakni dalam hal efisiensi waktu eksekusi ketika dihadapkan pada pencarian teks dalam skala besar. Tantangan ini muncul karena perlu dilakukannya evaluasi kecocokan berulang pada setiap generasi. Oleh karena itu, diusulkan sebuah pengoptimalan algoritma genetika dengan *multithreading* untuk pencarian kalimat. Dengan mengembangkan pengoptimalan algoritma genetika dengan *multithreading* untuk pencarian kalimat, diharapkan agar dapat lebih cepat dan efisien dalam pencarian kalimat.

2. METODE PENELITIAN

Pada sub-bab ini, diberikan penjelasan singkat mengenai tahapan penelitian serta pendekatan yang diterapkan dalam penelitian yang dilakukan dengan tujuan meningkatkan efisiensi algoritma genetika dalam pencarian kalimat. Metode penelitian yang digunakan berfokus pada menggabungkan algoritma genetika dengan pemrosesan multithreading untuk mengatasi masalah kinerja waktu eksekusi.

2.1 Sistem Secara Umum

Pada penelitian ini dilakukan implementasi dari perancangan dan desain sistem yang sudah dibuat, berupa kode program dengan *multithreading* dan menggunakan *single-threading*. Sistem ini menerapkan algoritma genetika standar untuk mewakili populasi dan melakukan operasi-operasi genetikanya. Perbedaannya proses evaluasi *fitness* pada setiap individu didistribusikan ke beberapa *thread* secara paralel. Setiap *thread* bertanggung jawab mengevaluasi individu tertentu. Dengan demikian, proses evaluasi *fitness* yang merupakan tahap paling lama dapat dieksekusi secara paralel, sehingga diharapkan dapat mempercepat proses evolusi dan meningkatkan efisiensi waktu eksekusi keseluruhan untuk mencari kalimat yang diberikan. Kemudian melakukan perbandingan kinerja waktu antara algoritma genetika asli dan algoritma genetika yang dioptimalkan dengan *multithreading* dalam pencarian. Pada penelitian ini dilakukan percobaan dengan menggunakan 30 buah *data uji* berupa 30 buah kalimat berbeda yang akan diujicobakan dengan menggunakan 3 skenario pengujian untuk menguji algoritma genetika dan multithreading. Sehingga perbedaan kinerja waktu antara algoritma genetika asli dan algoritma genetika yang dioptimalkan dengan *multithreading*. Setelah itu menghitung waktu eksekusi algoritma yang masing-masing dievaluasi tersendiri secara paralel.

2.2 Algoritma Genetika

Salah satu teknik yang didasarkan pada teori Darwin untuk menentukan optimalisasi adalah algoritma genetika. Prosedur algoritma ini dimulai dengan menentukan set solusi yang mungkin dan menggunakan algoritma genetika untuk melakukan perubahan dengan berulang kali (iterasi) untuk menghasilkan solusi terbaik. Kromosom adalah kelompok solusi potensial yang

telah didefinisikan sejak awal. Kromosom ini terdiri dari kumpulan angka biner yang dipilih secara acak. Semua set kromosom yang ditemukan adalah populasi.[6].

Generasi adalah tahap iterasi di mana kromosom berevolusi beberapa kali. Proses kawin silang (*crossover*) dan mutasi (*mutation*) menghasilkan generasi baru. *Crossover* melibatkan pemisahan atau pemisahan dua kromosom, kemudian menggabungkan setengah dari masing-masing kromosom dengan pasangan kromosom lain.[1]. Satu bit atau bagian kromosom ditransfer ke bagian lain dari kromosom pasangan. Kriteria kesesuaian juga dikenal sebagai kesesuaian kemudian digunakan untuk mengembangkan kromosom-kromosom tersebut. Hasil terbaik akan dipilih sedangkan yang lainnya diabaikan. Proses ini dilakukan berulang kali sampai menemukan kromosom dengan kesesuaian terbaik (*fitness* terbaik) untuk menyelesaikan masalah.[7].

Algoritma genetika terdiri dari delapan bagian utama, yaitu:

a. Inisialisasi
   Suatu proses yang menghasilkan sejumlah individu secara acak. Jenis operator genetika yang akan digunakan dan masalah yang akan diselesaikan akan menentukan banyaknya populasi. Setelah populasi dihitung, kromosom diinisialisasi secara acak. Ini dilakukan dengan mempertimbangkan domain solusi dan kendala masalah.[6].
b. Evaluasi
   Individu dinilai berdasarkan fungsinya. Setiap individu yang memiliki nilai *fitness* tinggi pada kromosomnya akan dipertahankan, sedangkan masing-masing individu yang memiliki nilai *fitness* rendah akan diganti. Fungsi *fitness* bergantung pada masalah representasi yang digunakan[8].
c. Seleksi
   Tujuan dari proses seleksi adalah untuk memilih individu yang akan dipilih untuk proses persilangan dan mutasi, yang akan menghasilkan calon induk yang berkualitas tinggi. Cari nilai *fitness* dengan langkah pertama dalam proses seleksi[9] Probabilitas reproduksi setiap individu dalam wadah seleksi tergantung pada nilai objektif dirinya sendiri dibandingkan dengan nilai objektif semua individu dalam wadah seleksi tersebut. Pada tahap seleksi berikutnya, nilai *fitness* inilah yang akan digunakan.[10].
d. *Roulette wheel selection*
   Seleksi ini dilakukan dengan memilih *parent* untuk mempertahankan nilai *fitness*-nya, sehingga memiliki kesempatan untuk memilih kromosom yang tepat. Semua kromosom ditempatkan dalam populasi sesuai dengan fungsi *fitness*, mirip dengan permainan roda rolet.[11]. Kromosom dipilih berdasarkan nilai *fitness*. Nilai *fitness* yang lebih tinggi meningkatkan peluang kromosom untuk dipilih berulang. Metode seleksi roda rolet, juga disebut sebagai seleksi *stochastik sampling with replacement* dengan penggantian, adalah yang paling sederhana.[12].
e. Seleksi *good fitness*
   Populasi induk adalah hasil dari seleksi di mana setengah dari populasi dengan nilai *fitness* terendah dihilangkan, sehingga hanya sekelompok solusi terbaik yang tersisa.[9]. Solusi baru harus dibuat sebanyak setengah dari populasi saat ini karena populasi harus tetap. Reproduksi kromosom baru dan mutasi dari solusi induk adalah dua metode yang digunakan untuk menghasilkan solusi baru. Tujuan pembuatan solusi baru ini adalah untuk menemukan solusi alternatif yang lebih baik daripada yang sudah ada.[13].
f. *Crossover* (Persilangan)

   Crossover adalah dapat menggabungkan dua kromosom ayah dan ibu sekaligus menjadi kromosom baru[14]. Proses *crossover* dapat menghasilkan kromosom yang menghasilkan solusi terbaik.

g. *Mutation* (Mutasi)

Mengodifikasi salah satu atau lebih gen dari sebuah kromosom. Ini dilakukan untuk menggantikan gen yang hilang dari populasi karena proses seleksi, yang memungkinkan gen yang tidak muncul kembali pada inisialisasi populasi[15].

h. Parameter Kontrol
Untuk mengontrol operator-operator seleksi, parameter kontrol genetika diperlukan. Kinerja algoritma genetika dalam memecahkan ditentukan oleh pemilihan parameter genetika. Algoritma genetika memiliki dua parameter utama: probabilitas *crossover* (Pc) dan probabilitas mutasi (Pm). Mutasi jarang terjadi pada seleksi alam murni, sehingga operator mutasi tidak selalu terjadi pada algoritma genetik. Nilai kemungkinan mutasi yang disarankan adalah sekitar 0,5% hingga 1%.[7]

2.3 Multithreading

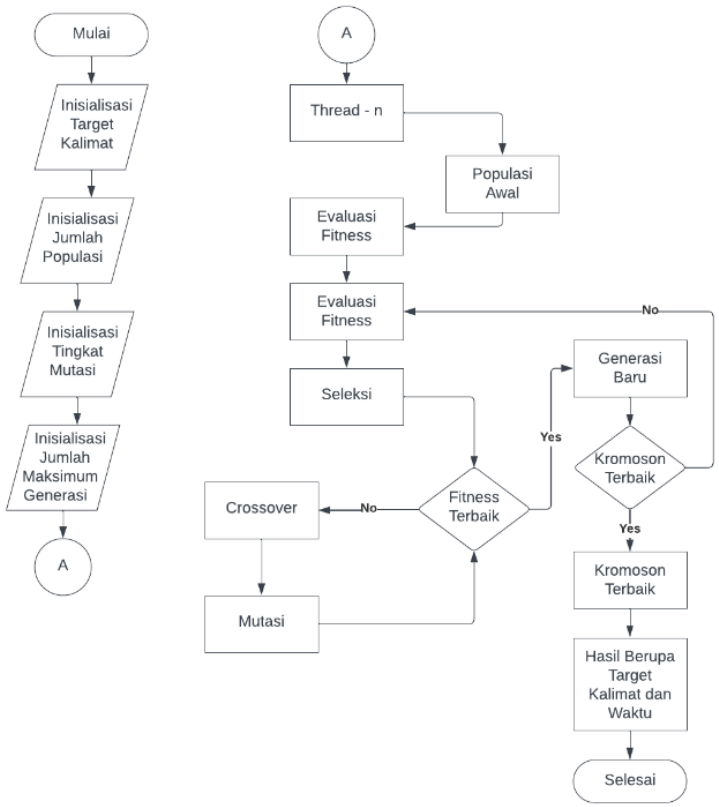
Teknik pemrograman yang dikenal sebagai *multithreading* memungkinkan beberapa *subproses* dalam program berjalan secara bersamaan. Sebuah *thread* adalah komponen program yang dapat berjalan sendiri, memungkinkan dua atau lebih *thread* berjalan bersamaan tanpa menunggu yang satu selesai. Penggunaan *multithread* meningkatkan efektivitas sumber daya[5] mengizinkan *thread* berjalan pada *core* prosesor yang berbeda, memungkinkan peningkatan potensi komputasi prosesor *multicore*. Jumlah *core* dalam sistem komputer sangat memengaruhi kinerja *multithread* karena *core* adalah unit pemrosesan CPU utama dan setiap inti memiliki kemampuan untuk menjalankan instruksi CPU secara paralel[16]. Gambar 1 merupakan potongan kode program implementasi multithreading untuk menghitung *crossover* dan mutasi dalam sistem pencarian kata menggunakan algoritma genetika secara paralel.

```
    # Menggunakan multithreading untuk menghitung crossover dan mutasi secara paralel
    threads = []
    new_population = []

    for p1, p2 in parents:
        thread = threading.Thread(
            target=lambda p1=p1, p2=p2: new_population.append(mutate(crossover(p1, p2), mutation_rate)))
        threads.append(thread)
        thread.start()

    for thread in threads:
        thread.join()

    population = new_population

else:
    print(
        f"Target-target tidak ditemukan dalam {max_generations} generasi.")
```

Gambar 1. Implementasi Multithreading untuk Menghitung Crossover dan Mutasi.

Prosedur umum untuk *multithreading* sebagai berikut; Pertama mulai dengan menginisialisasi *thread* yang digunakan program. Selanjutnya, sistem operasi atau lingkungan *runtime* akan mengeksekusi *thread* secara bersamaan. *Thread* mungkin perlu berinteraksi satu sama lain atau berbagi sumber daya umum seperti memori dalam beberapa situasi. Ada kemampuan untuk berkomunikasi satu sama lain melalui variabel bersama atau mekanisme lain yang ditawarkan oleh *library multithreading* atau bahasa pemrograman. Semua *thread* melakukan tugas sesuai dengan yang diberikan, dan ketika tugas selesai, *thread* dapat berhenti atau tetap aktif untuk melakukan tugas lain. Jadi, pengoptimalan algoritma genetika untuk pencarian kalimat menjadi lebih cepat dan efisien dengan *multithreading*.[17]. Selanjutnya dapat dilakukan

perhitungan waktu eksekusi setiap proses yang dijalankan. Gambar 2 merupakan diagram alir perancangan perangkat lunak pengoptimalan algoritma genetika dengan *multithreading*.



Gambar 2 Diagram Alir Pengoptimalan Algoritma Genetika Dengan *Multithreading*

3. HASIL DAN PEMBAHASAN

Pada bagian ini dijelaskan implementasi sistem yang digunakan algoritma genetika untuk menemukan solusi terhadap satu atau lebih kalimat target. Di mana implementasi sistem dalam kode yang menggunakan *multithreading* dan implementasi sistem tanpa menggunakan *multithreading*. Proses menggunakan inisialisasi ukuran populasi sebanyak 100, tingkat mutasi sebesar 0.1, dan jumlah maksimum generasi sebanyak 10.000. Dari target kalimat dari 1 – 30, dilakukan 3 skenario pengujian dengan menggunakan *single-threading*, menggunakan *multithreading* 2 *core*, dan menggunakan *multithreading 4 core*.

Pada penerapan pengoptimalan algoritma genetika dengan *multithreading* untuk pencarian kalimat, digunakan 3 skenario percobaan berbeda diuji yaitu tanpa *multithreading*, *multithreading* 2 *core*, dan *multithreading* 4 *core* sebanyak 30 kali percobaan. Fokus utama adalah pada waktu eksekusi dan rata-rata waktu eksekusi dalam setiap konfigurasi. Penggunaan *multithreading* adalah strategi yang umum digunakan untuk memaksimalkan penggunaan sumber daya prosesor, terutama dalam menghadapi aplikasi yang membutuhkan pemrosesan tugas secara paralel. Hasil analisis data menunjukkan perbandingan yang menarik antara penggunaan *multithreading* dan tanpa *multithreading*. Ketika mengamati konfigurasi dengan *multithreading* pada 2 *core*, bahwa meskipun ada sedikit peningkatan dalam kinerja dibandingkan tanpa

multithreading, perbedaannya sangat kecil dan mungkin tidak memberikan manfaat yang signifikan dalam pengoptimalan algoritma genetika. Hasil ini menyoroti perlunya pertimbangan yang cermat dalam alokasi sumber daya prosesor dan pengoptimalan. Gambar 3 merupakan proses pengujian dan hasil eksekusi pencarian sembarang kalimat menggunakan *singlethreading.*

```
PS D:\Py> & "C:/Program Files/Python311/python.exe" d:/Py/test/ag_cari.py
Masukkan target kalimat (pisahkan dengan koma): bakso
Masukkan ukuran populasi: 100
Masukkan tingkat mutasi (0-1): 0.1
Masukkan jumlah maksimum generasi: 1000
Generasi 1: ['bdsyo'] (fitness: [2])
Generasi 2: ['baiso'] (fitness: [4])
Generasi 3: ['baezo'] (fitness: [3])
Generasi 4: ['bmkso'] (fitness: [4])
Generasi 5: ['bdkso'] (fitness: [4])
Generasi 6: ['bakso'] (fitness: [5])
Semua target ditemukan dalam 6 generasi.
Waktu eksekusi program: 14.06 detik
```

Gambar 3. Hasil Eksekusi Menggunakan *Singlethreading*

Gambar 4 dan Gambar 5 merupakan hasil eksekusi pencarian sembarang kalimat dengan menggunakan *multithreading.* Pada pengujian multithreading dilakukan hal yang sama seperti singlethreading, yaitu eksekusi program berbeda pada pencarian keyword.

```
PS D:\Py> & "C:/Program Files/Python311/python.exe" d:/Py/test/ag_multi.py
Masukkan target kalimat (pisahkan dengan koma): bakso
Masukkan ukuran populasi: 100
Masukkan tingkat mutasi (0-1): 0.1
Masukkan jumlah maksimum generasi: 1000
Generasi 1: ['atkgk'] (fitness: [1])
Generasi 2: ['atkko'] (fitness: [2])
Generasi 3: ['tawso'] (fitness: [3])
Generasi 4: ['banzo'] (fitness: [3])
Generasi 5: ['babso'] (fitness: [4])
Generasi 6: ['bagso'] (fitness: [4])
Generasi 7: ['babso'] (fitness: [4])
Generasi 8: ['bakgo'] (fitness: [4])
Generasi 9: ['bakso'] (fitness: [5])
Semua target ditemukan dalam 9 generasi.
Waktu eksekusi program: 13.18 detik
```

Gambar 4. Hasil Eksekusi Menggunakan *Multithreading 2 Core*

```
Masukkan target kalimat (pisahkan dengan koma):  bakso
Masukkan ukuran populasi:  100
Masukkan tingkat mutasi (0-1):  0.1
Masukkan jumlah maksimum generasi:  1000
Generasi 1: ['bxkaz'] (fitness: [2])
Generasi 2: ['bauqm'] (fitness: [2])
Generasi 3: ['baktl'] (fitness: [3])
Generasi 4: ['nawso'] (fitness: [3])
Generasi 5: ['naksn'] (fitness: [3])
Generasi 6: ['baksn'] (fitness: [4])
Generasi 7: ['baksy'] (fitness: [4])
Generasi 8: ['bakso'] (fitness: [5])
Semua target ditemukan dalam 8 generasi.
Waktu eksekusi program: 6.81 detik
```

Gambar 5. Hasil Eksekusi Menggunakan *Multithreading 4 Core*

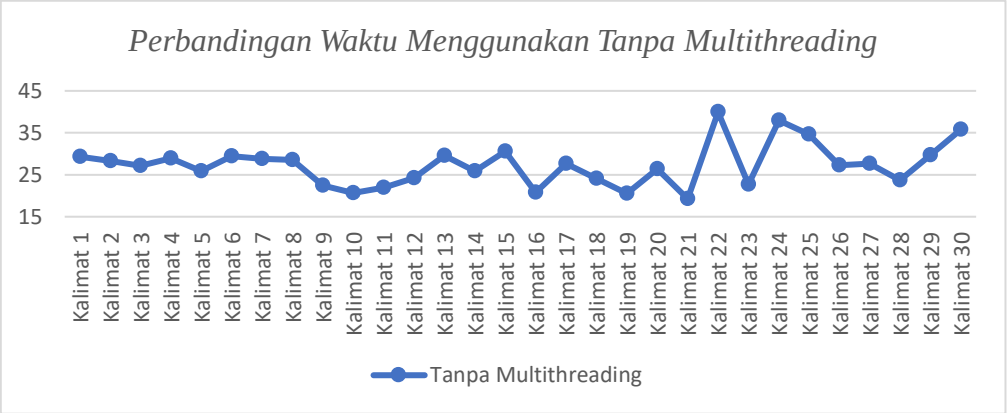
Selanjutnya dilakukan pengujian sistem dengan menggunakan 3 skenario percobaan yaitu menggunakan *singlethreading* (tanpa menggunakan *multithreading*), menggunakan

multithreading 2 core, dan menggunakan *multithreading 4 core*. Data yang digunakan dalam pengujian sistem berupa 30 keyword. Hasil pengujian tanpa menggunakan *multithreading*, *multithreading* 2 core, dan *multithreading* 4 core dapat dilihat pada Tabel 1.

Tabel 1 Hasil Pengujian Sistem

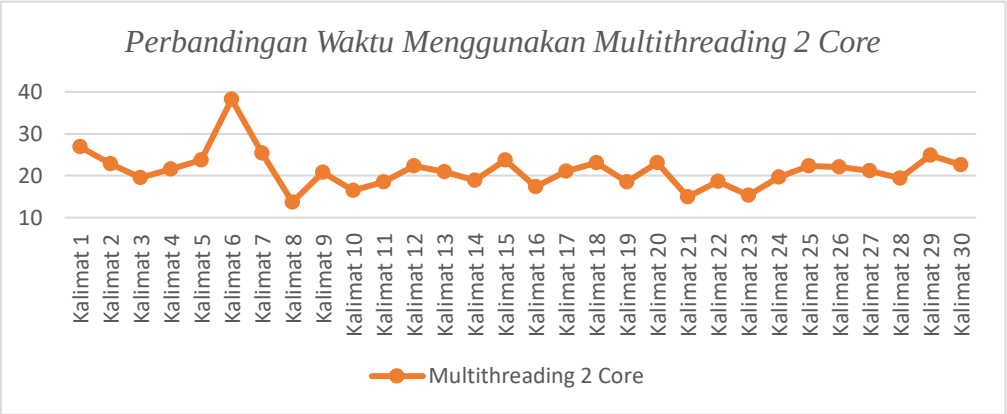
No.	Kalimat	Waktu		
		Tanpa Menggunakan *Multithreading*	Menggunakan *Multithreading 2 Core*	Menggunakan *Multithreading 4 Core*
1	Bunga indah	29.28 s	26.91 s	21.70 s
2	Hujan ringan	28.29 s	22.83 s	19.30 s
3	Senyum manis	27.15 s	19.55 s	14.34 s
4	Pesawat terbang	28.92 s	21.52 s	18.71 s
5	Matahari terbenam	25.90 s	23.66 s	17.38 s
6	Pohon tinggi	29.40 s	38.21 s	21.34 s
7	Laut biru	28.85 s	25.39 s	19.76 s
8	Buku tebal	28.49 s	13.69 s	12.88 s
9	Anjing lucu	22.38 s	20.82 s	18.25 s
10	Kota besar	20.68 s	16.41 s	14.32 s
11	Gitar akustik	21.89 s	18.54 s	16.33 s
12	Pakaian modis	24.26 s	22.31 s	19.02 s
13	Rumah kecil	29.54 s	20.97 s	13.25 s
14	Sungai mengalir	25.84 s	18.83 s	12.83 s
15	Kucing menggemaskan	30.60 s	23.68 s	18.29 s
16	Kafe ramai	20.79 s	17.35 s	11.46 s
17	Pagi cerah	27.62 s	21.09 s	14.30 s
18	Pelukan hangat	24.03 s	23.03 s	11.29 s
19	Selimut lembut	20.48 s	18.46 s	7.16 s
20	Pantai berpasir	26.39 s	23.13 s	12.58 s
21	Awan berawan	19.25 s	14.93 s	13.87 s
22	Laut tenang	39.94 s	18.61 s	13.61 s
23	Malam tenang	22.69 s	15.37 s	12.91 s
24	Senja cantik	37.96 s	19.65 s	13.89 s
25	Perjalanan jauh	34.59 s	22.29 s	16.76 s
26	Bintang bersinar	27.23 s	22.03 s	19.92 s
27	Pulau eksotis	27.70 s	21.23 s	18.05 s
28	Hutan lebat	23.77 s	19.39 s	15.19 s
29	Air terjun tinggi	29.72 s	24.89 s	12.93 s
30	Rumah mewah	35.76 s	22.60 s	20.67 s

Dari data tabel 1 tersebut waktu menggunakan tanpa *Multithreading* kemudian diolah dan direpresentasikan dalam bentuk grafik, seperti yang terlihat pada Gambar 6.



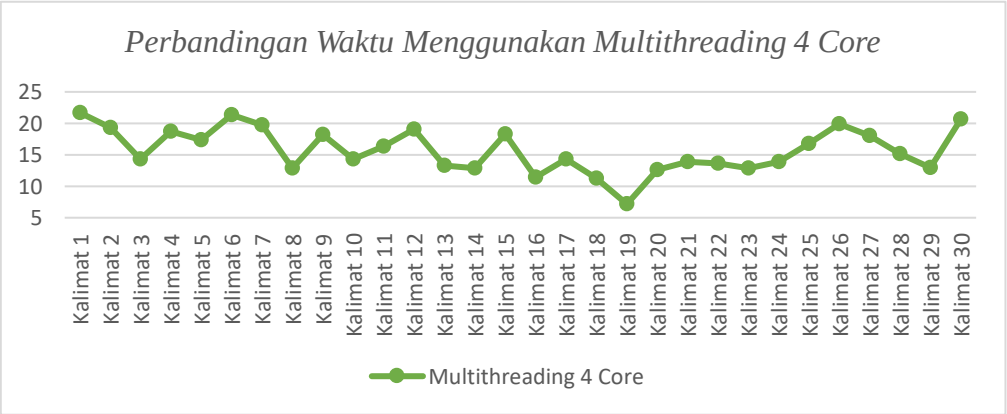
Gambar 6. Perbandingan Waktu Menggunakan Tanpa *Multithreading*

Dari data Tabel 1 tersebut waktu menggunakan *Multithreading 2 Core* kemudian diolah dan direpresentasikan dalam bentuk grafik, seperti yang terlihat pada Gambar 7.



Gambar 7. Perbandingan Waktu Menggunakan *Multithreading 2 Core*

Dari data tabel 1 tersebut waktu menggunakan *Multithreading 4 Core* kemudian diolah dan direpresentasikan dalam bentuk grafik, seperti yang terlihat pada Gambar 8.



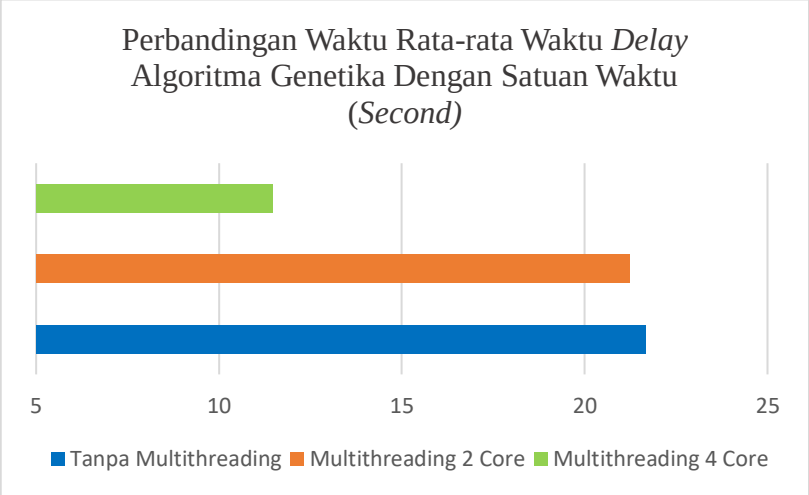
Gambar 8. Perbandingan Waktu Menggunakan *Multithreading 4 Core*

Nilai Rata - rata *delay* sistem algoritma genetika merupakan nilai yang menggambarkan rata-rata waktu *delay* (waktu tunggu) dalam satuan detik pada algoritma genetika. Tabel ini dibagi menjadi tiga bagian yang mencakup berbagai skenario, yaitu "Dengan Tanpa Menggunakan *Multithreading* " dengan rata-rata *delay* 21,66 detik, "Dengan Menggunakan *Multithreading 2 core* " dengan rata-rata *delay* 21,24 detik, dan " Dengan Menggunakan *Multithreading 4 core* " dengan rata-rata *delay* 11,48 detik. Rata-rata *delay* algoritma genetika dapat dilihat pada Tabel 2.

Tabel 2 Rata-Rata Delay Algoritma Genetika

Skenario	**∑(x1,x30) Delay**	**Total**
Tanpa *Multithreading*	649.89 s	21.66 s
Menggunakan *Multithreading 2 core*	637.37 s	21.24 s
Menggunakan *Multithreading 4 core*	344.44 s	11.48 s

Dari data tabel 2 tersebut rata-rata waktu *delay* algoritma genetika kemudian diolah dan direpresentasikan dalam bentuk grafik, seperti yang terlihat pada Gambar 9.



Gambar 9. Perbandingan Waktu Rata-rata Waktu *Delay* Algoritma Genetika

Berdasarkan hasil pengujian yang dilakukan, dapat dilihat bahwa penggunaan *multithreading* pada 2 *core* dapat meningkatkan kinerja pencarian kata jika dibandingkan dengan tanpa *multithreading*, yaitu mendapatkan nilai selisih waktu sebesar 0,42 detik. Waktu eksekusi hampir sama, meskipun ada sedikit peningkatan dalam rata-rata waktu eksekusi. Penggunaan *multithreading* pada 4 *core* menghasilkan peningkatan kinerja yaitu sekitar 10,18 detik jika dibandingkan dengan waktu eksekusi tanpa multithreading, dan 9,76 detik jika dibandingkan dengan waktu eksekusi menggunakan multithreading 2 core. Waktu eksekusi menjadi hampir separuh dari yang digunakan tanpa *multithreading*, dan rata-rata waktu eksekusi lebih dari dua kali lebih cepat. Dengan demikian, untuk meningkatkan kinerja aplikasi, disarankan untuk menggunakan *multithreading* dengan 4 *core*, karena hasilnya jauh lebih baik dibandingkan dengan konfigurasi lainnya. Pada konfigurasi *multithreading* 4 *core*, data menunjukkan peningkatan kinerja yang signifikan dan waktu eksekusi berkurang hampir setengahnya jika dibandingkan dengan penggunaan tanpa *multithreading*. Ini menunjukkan bahwa penggunaan *multithreading* pada jumlah *core* yang lebih besar dapat secara signifikan meningkatkan kinerja aplikasi, terutama ketika tugas-tugas dapat dieksekusi secara paralel. Keputusan pemilihan konfigurasi *multithreading* harus mempertimbangkan tujuan aplikasi dan karakteristik sumber daya yang tersedia.

4. KESIMPULAN

Kesimpulan yang dapat diambil dari penelitian ini sebagai berikut:

a. Mengoptimalkan kinerja algoritma genetika untuk pencarian kalimat dapat dicapai dengan membagi populasi individu menjadi beberapa *subset* yang dievaluasi secara paralel oleh masing-masing *thread* Proses evaluasi *fitness* yang merupakan tahapan paling lama dapat dieksekusi secara paralel oleh *thread-thread* Semakin banyak jumlah *core* CPU yang digunakan (2 core, dan 4 core), semakin cepat proses evaluasi karena pekerjaan terdistribusi.
b. Perbedaan kinerja waktu antara algoritma genetika asli dan yang dioptimalkan dengan *multithreading* terletak pada efisiensi penggunaan sumber daya komputasi. Tanpa *multithreading* membutuhkan waktu rata-rata 21,66 detik per generasi Dengan 2 *core multithreading* menjadi 21,24 detik per generasi Dengan 4 *core multithreading* menjadi 11,48 detik per generasi Dengan kata lain, implementasi *multithreading* dapat meningkatkan kecepatan eksekusi algoritma hingga 50% karena proses *fitness* dievaluasi secara paralel. Algoritma yang dioptimalkan dengan *multithreading*, khususnya dengan 4 *core*, memberikan waktu eksekusi yang jauh lebih singkat dibandingkan algoritma asli.

5. SARAN

Berikut beberapa saran yang dapat diambil dari hasil pengujian yang telah dilaksanakan antara lain:

a. Penentuan ukuran populasi yang tepat sesuai jumlah *thread* agar pekerjaan terdistribusi dengan baik. Semakin besar populasi, semakin banyak data yang dapat diproses secara paralel.
b. Pengujian dengan berbagai konfigurasi jumlah thread untuk mendapatkan optimumnya sesuai spesifikasi hardware.

DAFTAR PUSTAKA

[1] A. Rahman, E. Utami, and S. Sudarmawan, "Sentimen Analisis Terhadap Aplikasi pada Google Playstore Menggunakan Algoritma Naïve Bayes dan Algoritma Genetika," *Jurnal Komtika (Komputasi dan Informatika)*, vol. 5, no. 1, pp. 60–71, Jul. 2021, doi: 10.31603/komtika.v5i1.5188.

[2] R. Afira and R. Wijaya, "Penjadwalan Mata Pelajaran dengan Algoritma Genetika (Studi Kasus di SMK Negeri 1 Padang)," *Jurnal KomtekInfo*, vol. 8, no. 2, pp. 140–144, Jun. 2021, doi: 10.35134/komtekinfo.v8i2.109.

[3] Rejoice Iboy Erwin Saragih, "Penerapan Algoritma Genetika Pada Pengenalan Paragraf," *Journal Information System Development (ISD)* , vol. 4, 2019.

[4] M. Fachrie and Anita Fira Waluyo, "Model Paralelisasi Algoritma Genetika Terpandu pada Sistem Penjadwalan Kuliah Universitas dengan Alokasi Waktu Dinamis," *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)*, vol. 5, no. 3, pp. 550–556, Jun. 2021, doi: 10.29207/resti.v5i3.2988.

[5] Megah Mulya, "Penerapan Multi-threading untuk Meningkatkan Kinerja Pengolahan Citra Digital," *Jurnal Generic*, vol. 8, 2013.

[6] D. Setiawan, R. N. Putri, and R. Suryanita, "Perbandingan Algoritma Genetika dan Backpropagation pada Aplikasi Prediksi Penyakit Autoimun," *Khazanah Informatika : Jurnal Ilmu Komputer dan Informatika*, vol. 5, no. 1, pp. 21–27, Jun. 2019, doi: 10.23917/khif.v5i1.7173.

[7] M. F. Azim, A. Azizah, and D. Anggraini, "Optimasi Bobot Portofolio Menggunakan Algoritma Genetika," *Jurnal Sains Matematika dan Statistika*, vol. 7, no. 1, p. 58, Mar. 2021, doi: 10.24014/jsms.v7i1.12190.

[8] D. M. Utama, L. R. Ardiansyah, and A. K. Garside, "Penjadwalan Flow Shop untuk Meminimasi Total Tardiness Menggunakan Algoritma Cross Entropy–Algoritma Genetika," *Jurnal Optimasi Sistem Industri*, vol. 18, no. 2, pp. 133–141, Oct. 2019, doi: 10.25077/josi.v18.n2.p133-141.2019.

[9] M. Nur Cahya, I. Elan Maulani, I. Intan, and T. Ayu Ambarwati, "Penerapan Algoritma Genetika dalam Optimisasi Penjadwalan Sistem Informasi Akademik," *Jurnal Sosial Teknologi*, vol. 3, no. 2, pp. 103–107, Feb. 2023, doi: 10.59188/jurnalsostech.v3i2.637.

[10] D. Cahya Putri Buani, "Penerapan Algoritma Naïve Bayes dengan Seleksi Fitur Algoritma Genetika Untuk Prediksi Gagal Jantung," *EVOLUSI : Jurnal Sains dan Manajemen*, vol. 9, no. 2, Sep. 2021, doi: 10.31294/evolusi.v9i2.11141.

[11] Entot Suhartono, "Optimasi Penjadwalan Mata Kuliah Dengan Algoritma Genetika (Studi Kasus Di Amik Jtc Semarang)," *INFOKAM*, vol. 11, 2015.

[12] P. Arsi and O. Somantri, "Deteksi Dini Penyakit Diabetes Menggunakan Algoritma Neural Network Berbasiskan Algoritma Genetika," *Jurnal Informatika: Jurnal Pengembangan IT*, vol. 3, no. 3, pp. 290–294, Oct. 2018, doi: 10.30591/jpit.v3i3.1008.

[13] A. Assagaf, A. Ibrahim, and C. Suranto, "Membangun Sistem Informasi Penjadwalan Dengan Metode Algoritma Genetika Pada Laboratorium Teknik Informatika Universitas Muhammadiyah Maluku Utara," *Jurnal Ilmiah ILKOMINFO - Ilmu Komputer & Informatika*, vol. 1, no. 2, Jul. 2018, doi: 10.47324/ilkominfo.v1i2.13.

[14] A. Riandana, "Penjadwalan Mata Pelajaran dengan Algoritma Genetika (Studi Kasus di SMK Negeri 1 Padang)," *Jurnal KomtekInfo* , vol. 8, 2021.

[15] D. Cahya Putri Buani, "Penerapan Algoritma Naïve Bayes dengan Seleksi Fitur Algoritma Genetika Untuk Prediksi Gagal Jantung," *EVOLUSI : Jurnal Sains dan Manajemen*, vol. 9, no. 2, Sep. 2021, doi: 10.31294/evolusi.v9i2.11141.

[16] Mohamad Firdaus, "Perancangan aplikasi chat-room dengan prinsip threading melalui pemrograman dengan bahasa java," *TEKNOSAINS : Jurnal Sains, Teknologi dan Informatika*, vol. 9, no. 2, pp. 121–135, Jul. 2022, doi: 10.37373/tekno.v9i2.242.

[17] A. Syauqi and A. N. Hidayah, "Implementasi Komputasi Paralel untuk Optimalisasi Komputasi Pada Aplikasi Transliterasi Huruf Latin ke Aksara Jawa," *Jurnal Online Informatika*, vol. 3, no. 1, p. 29, Jun. 2018, doi: 10.15575/join.v3i1.179.