Analisis Kompleksitas Algoritma BLS dan TLS dalam Deteksi Exoplanet Berbasis Data Light Curve

Athian Nugraha Muarajuang - 13523106[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesa 10 Bandung 40132, Indonesia
[1]athianbintang@gmail.com, 13523106@std.stei.itb.ac.id

***Abstract*—Metode transit telah menjadi salah satu pendekatan utama dalam mendeteksi eksoplanet, dengan mengamati peredupan cahaya bintang secara berkala yang disebabkan oleh planet yang melintas di depannya. Dalam makalah ini, dua algoritma utama dalam deteksi transit, yaitu Box Least Squares (BLS) dan Transit Least Squares (TLS), dibandingkan untuk mengevaluasi efisiensi dan akurasi keduanya berdasarkan kompleksitas waktu. BLS, yang menggunakan model transit berbentuk kotak, menawarkan kecepatan pemrosesan tinggi tetapi kurang sensitif terhadap transit kecil. Di sisi lain, TLS menggunakan model kurva transit U/V yang lebih realistis, memberikan sensitivitas lebih tinggi terhadap transit dangkal, namun dengan waktu eksekusi yang lebih besar. Hasil analisis menunjukkan bahwa BLS lebih efisien untuk dataset besar, sementara TLS lebih cocok untuk analisis yang lebih mendalam. Studi ini memberikan wawasan mengenai pemilihan algoritma yang sesuai dengan kebutuhan analisis data light curve.**

***Kata Kunci*—Deteksi Eksoplanet, Metode Transit, Box Least Squares, Transit Least Squares, Kompleksitas Algoritma.**

I. Pendahuluan

Pencarian eksoplanet melalui metode transit telah menjadi salah satu metode utama dalam mengidentifikasi planet di luar tata surya. Deteksi ini bergantung pada observasi peredupan cahaya bintang secara berkala akibat objek yang melintas di depannya. Algoritma pendeteksian transit telah berkembang dari pendekatan yang sederhana menjadi metode yang lebih kompleks dan akurat untuk menemukan eksoplanet kecil. Dalam konteks ini, algoritma seperti Box-fitting Least Squares (BLS) dan Transit Least Squares (TLS) memainkan peran kunci dalam analisis data light curve untuk mengidentifikasi pola transit.

Algoritma BLS didesain untuk menemukan pola transit dalam bentuk kotak pada data light curve. Prinsipnya adalah mendeteksi perubahan kecerlangan bintang yang teratur, seperti penurunan tajam diikuti oleh kembalinya kecerlangan ke nilai semula. Algoritma ini memiliki keunggulan dalam kemudahan implementasinya, sehingga sering digunakan dalam berbagai perangkat lunak astronomi. Namun, BLS dapat menghadapi tantangan dalam sensitivitasnya terhadap planet berukuran kecil atau transit berdurasi pendek, terutama saat menangani volume data yang sangat besar.

Algoritma TLS dikembangkan sebagai penyempurnaan dari BLS dengan mempertimbangkan model transit yang lebih akurat. Algoritma ini dimaksudkan untuk menjadi lebih sensitif terhadap transit yang disebabkan oleh planet kecil untuk mengurangi tingkat positif palsu. Dengan pendekatan ini, TLS dapat mendeteksi sinyal transit yang lebih halus, yang sering diabaikan oleh algoritma sebelumnya. TLS menjadi instrumen penting dalam analisis data dari misi seperti Kepler dan TESS yang berfokus pada eksoplanet berukuran kecil.

Makalah ini bertujuan untuk membahas analisis kompleksitas dua algoritma utama dalam deteksi transit, yaitu BLS dan TLS, guna memahami kelebihan dan kekurangannya dalam mendeteksi eksoplanet berdasarkan data light curve. Dengan mempertimbangkan efisiensi komputasi, sensitivitas terhadap data berkualitas rendah, serta keakuratan dalam mendeteksi planet kecil, penelitian ini diharapkan memberikan wawasan baru tentang efektivitas kedua algoritma tersebut. Hasil analisis juga diharapkan mampu mendukung pengembangan algoritma lebih efisien dan akurat di masa depan dalam mendukung eksplorasi eksoplanet.

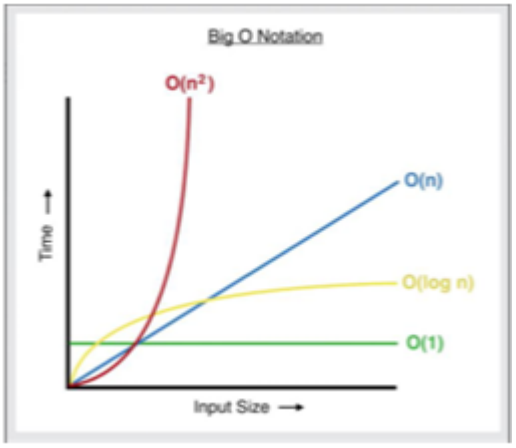
II. Landasan Teori

A. Kompleksitas Algoritma

Dalam analisis algoritma, kompleksitas adalah aspek krusial yang digunakan untuk mengukur efisiensi suatu algoritma dalam menyelesaikan masalah, baik dari segi waktu eksekusi (time complexity) maupun penggunaan ruang memori (space complexity). Compleksitas waktu, diwakili dengan notasi Big-O (O), mencerminkan hubungan antara waktu eksekusi algoritma dan ukuran input (N), sementara kompleksitas ruang mengindikasikan kebutuhan memori. Algoritma dengan kompleksitas waktu rendah seperti O(N) (linear) atau O(logN) (logaritmik) dianggap efisien dan dapat diandalkan untuk menangani masukan besar. Sebaliknya, algoritma dengan kompleksitas tinggi seperti O(N^2) (kuadratik) atau O(2^N) (eksponensial) menjadi tidak praktis untuk data besar karena waktu eksekusi meningkat secara signifikan dengan pertambahan ukuran data.

Notasi Big-O (O) digunakan dalam deskripsi waktu eksekusi algoritma dalam skenario terburuk (worst-case), mengabaikan konstanta atau faktor kecil sehingga memberikan gambaran efisiensi algoritma secara teoritis. Pemahaman kompleksitas algoritma menjadi landasan penting untuk membandingkan efisiensi metode dalam penelitian berbasis data besar, seperti analisis transit eksoplanet. Algoritma harus mampu mengevaluasi data dalam jumlah besar secara efisien, dan

dalam konteks ini, algoritma dengan kompleksitas rendah lebih diutamakan untuk performa yang optimal.



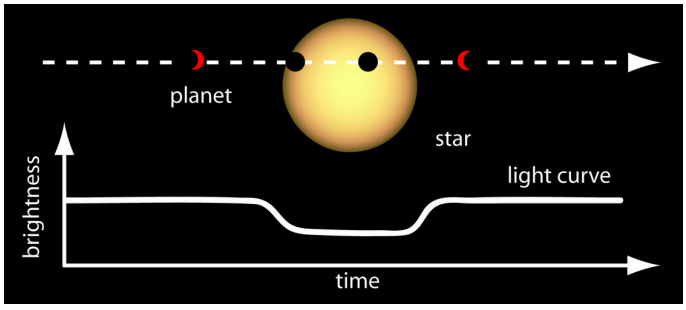
Gambar 1.1 Diagram Ilustrasi big-O
sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/25-Kompleksitas-Algoritma-Bagian1-2024.pdf

B. Deteksi Transit Eksoplanet

Deteksi eksoplanet dengan metode transit merupakan salah satu teknik yang populer digunakan dalam bidang astronomi untuk menemukan planet di luar Tata Surya. Dalam metode ini, penurunan kecerlangan bintang diukur ketika sebuah planet melewati depan bintang tersebut, terlihat dari sudut pandang pengamat menggunakan teleskop atau instrumen fotometri. Proses transit ini umumnya menghasilkan penurunan cahaya yang sangat kecil namun masih dapat diukur, bergantung pada perbandingan ukuran planet dan bintangnya. Semakin besar ukuran planet, maka penurunan kecerlangan yang teramati juga akan semakin signifikan.

Transit secara umum dapat dianggap sebagai fenomena periodik, dimana ketika planet menyelesaikan satu orbit, ia akan melintasi depan bintang pada saat yang hampir sama, menghasilkan tanda berulang dalam kurva cahaya. Melalui pola penurunan kecerlangan ini, kita dapat mengestimasi parameter-parameter kunci seperti:

1) Periode orbit, jarak waktu antara satu transit dengan transit berikutnya.
2) Kedalaman transit, perbedaan intensitas cahaya bintang sebelum dan saat transit terjadi, hal ini sangat berkaitan dengan rasio perbedaan ukuran planet dengan ukuran bintang.
3) Durasi transit, lama penurunan flux berlangsung, ini sangat dipengaruhi oleh geometri orbit yaitu inklinsi dan jari-jari orbit, juga ukuran bintang dan planet itu sendiri.



Gambar 1.2 Ilustrasi Transit Cahaya
sumber: www.nasa.gov

Untuk memantau transit, para astronom merekam perubahan kecerlangan bintang selama waktu yang konsisten dalam waktu yang konsisten. Dalam hal ini, dataset yang dihasilkan dalam deret waktu beserta nilai flux. Sebagian besar sumber gangguan stokastik (gangguan) memperburuk kondisi dengan menciptakan berbagai jenis kebisingan di sekitar transit mencari sinyal yang sangat lemah dan berumur pendek. Akibatnya, algoritma processing yang andal diperlukan untuk menemukan pola penurunan flux berulang kekerasan bocor di tengah kebisingan.

C. Box Least Square (BLS)

Algoritma BLS adalah alat khusus yang digunakan untuk menemukan planet yang mengorbit bintang jauh menggunakan pengukuran cahaya. Teknik ini dibuat oleh Kovács dkk. (2002) sebagai instrumen yang efektif dan mudah untuk mengamati grafik fotometrik yang menunjukkan indikasi okultasi. "Kovács dkk. (2002) merekayasa perangkat yang andal dan bijaksana untuk memeriksa rangkaian cahaya yang menandakan ecli planet 'Kotak Kuadrat Terkecil' menunjukkan metode utama algoritma ini, yang melibatkan pemodelan sinyal transit melalui konfigurasi kotak fundamental, yang dicirikan oleh bintang yang tiba-tiba mengalami pengurangan fluks selama interval tertentu sebelum pulih pasca transit.

Algoritma BLS bekerja dengan cara:

1. Data Kurva Cahaya Lipat (Folding).

Lipat data kecerahan menjadi waktu tunggu potensial bagi sebuah planet untuk melintasi bintang kita untuk melihat penurunan kecerahan secara teratur. BLS membuat angka lebih mudah dipahami dengan mengubah bagian waktu untuk menyembunyikan perubahan yang tidak berasal dari pola cahaya. Proses ini diulangi untuk setiap kandidat periode dalam rentang yang ditentukan.

2. Mencocokkan Model Transit

Setelah data dilipat, program komputer mencari penurunan pola persegi yang menunjukkan sedang terjadi perjalanan luar angkasa. Model kotak menggambarkan penurunan arus yang tajam dan tajam melalui rentang transit tertentu. Setelah itu, metode ini mencari tahu seberapa cocok model transit dengan data yang dikumpulkan.

3. Menghasilkan Periodogram

Hasil pemeriksaan untuk setiap interval pelamar ditampilkan dalam bentuk plot spektrum, yang menunjukkan amplitudo energi di seluruh interval yang diuji. Puncak dominan pada grafik frekuensi menunjukkan istilah yang paling mungkin menampung peristiwa persilangan bintang. Dari titik ini, variabel seperti waktu lintasan, panjang lintasan, dan kedalaman lintasan dapat dihitung.

D. Transit Least Square (TLS)

Transit Least Squares (TLS) merupakan algoritma yang dirancang untuk meningkatkan akurasi deteksi transit exoplanet dibandingkan dengan algoritma Box Least Squares

(BLS). TLS menggunakan model transit berbentuk U/V yang lebih realistis dan sesuai dengan teori astrofisika seperti penggelapan anggota tubuh, yang menggambarkan peredupan tepi bintang akibat temperatur atmosfer yang lebih rendah di area tersebut. Diperkenalkan oleh Hippke dan Heller (2019), TLS berfungsi sebagai alat yang lebih sensitif untuk mendeteksi transit berdurasi pendek atau transit dengan sinyal yang sangat halus.

Algoritma TLS bekerja dengan cara:

1. Pemodelan Transit Realistis

Pemodelan kurva cahaya transit yang memodelkan efek pelensaan yang kuat direpresentasikan dengan menggunakan profil kurva U/V yang detail dengan penggelapan anggota tubuh. Model ini mencakup faktor-faktor seperti ukuran relatif planet terhadap bintang induk, durasi transit, dan penggelapan anggota tubuh yang memang dapat terjadi di alam ketika sebuah planet transit pada sebuah bintang, dan menghasilkan model yang lebih efisien daripada model kotak dalam BLS.

2. Pencarian Periode

Algoritma memeriksa kandidat periode transit (P_k) dalam rentang yang telah ditentukan. Data *light curve* dilipat (*folded*) ke setiap kandidat periode menggunakan:

$$\phi = \left(\frac{t}{P_k} \right) \bmod 1$$

di mana ϕ adalah fase dalam periode yang diuji, dan data dilipat untuk mencari pola berulang dalam kurva cahaya.

3. Evaluasi Durasi Transit

Untuk setiap periode kandidat, algoritma ini mengevaluasi berbagai durasi transit (D) dengan menggunakan kisi durasi. Hal ini membuat TLS dapat membandingkan bentuk transit yang lebih fleksibel dibandingkan dengan algoritma lainnya.

4. Optimasi Least Squares

TLS menggunakan metode least squares untuk meminimalkan perbedaan antara data flux pengamatan (f_i) dan model flux transit ($f(t)$):

$$\chi^2 = \sum_{i=1}^{N} \left(f_i - f(t_i) \right)^2$$

di mana χ^2 menggambarkan kecocokan model transit dengan data.

5. Periodeogram Transit

TLS menghasilkan periodogram, yaitu kekuatan yang merupakan grafik kekuatan sinyal dalam kaitannya dengan periode kandidat. Puncak tertinggi dalam periodogram mengindikasikan periode yang paling mungkin mengandung transit.

III. Implementasi

Proses penerapan algoritma Box Least Squares (BLS) dan Transit Least Squares (TLS) dilakukan melalui implementasi dalam bahasa pemrograman Python. Pemilihan lingkungan pengembangan didasarkan pada kriteria fleksibilitas, ketersediaan pustaka saintifik, serta dukungan untuk pengelolaan data berbasis notebook. Google Colab dipilih sebagai platform utama karena bersifat berbasis cloud dan memiliki kemampuan untuk mempermudah eksperimen ilmiah.

Python 3.10 adalah versi yang digunakan, menjadi default di Google Colab pada tahun 2025. Pemilihan versi ini memastikan kompatibilitas dengan pustaka modern yang diperlukan dalam penelitian, termasuk NumPy, Matplotlib, Astropy, dan TransitLeastSquares. NumPy digunakan untuk operasi numerik dan manipulasi data array, sementara Matplotlib untuk visualisasi seperti light curve dan grafik analisis.

Astropy digunakan khususnya untuk algoritma BLS. BoxLeastSquares module dalam Astropy memungkinkan analisis light curve dengan metode box-fitting. Modul ini sekarang tersedia di astropy.timeseries pada versi terbaru, mempermudah pengolahan data sintetik maupun riil dalam skala besar. TransitLeastSquares digunakan untuk algoritma TLS, yang cenderung lebih spesifik dalam mendeteksi transit eksoplanet berdasarkan pola kurva cahaya planet. Pustaka ini mendukung pembersihan data otomatis untuk memastikan kelancaran analisis, meskipun ada data yang tidak lengkap atau mengandung noise.

Manajemen perpustakaan dilakukan secara langsung di dalam notebook dengan menggunakan perintah pip install untuk menginstal atau memperbarui perpustakaan yang diperlukan. Pemasangan perpustakaan dilakukan pada tahap awal pengembangan untuk memastikan kesesuaian versi yang digunakan. Sebagai contoh, berikut adalah perintah pemasangan perpustakaan TLS dan pembaruan perpustakaan Astropy:

```
!pip install transitleastsquares --quiet
!pip install --upgrade astropy --quiet
```

A. Deskripsi Data

Dalam penelitian ini, data dummy atau data sintetik digunakan untuk menguji algoritma Box Least Squares (BLS) dan Transit Least Squares (TLS). Penggunaan data dummy terpilih karena memberikan fleksibilitas penuh dalam mengendalikan karakteristik light curve, seperti periode transit, kedalaman transit, durasi transit, dan tingkat kebisingan (noise). Data ini direkayasa dengan pola flux dasar sebesar 1.0, yang menggambarkan kecerlangan bintang tanpa gangguan, dan kemudian disisipkan pola transit berbentuk box sederhana. Sebagai contoh, flux menurun sebesar 1% (kedalaman transit) setiap 2,5 hari (periode transit) dengan durasi transit 0,05 hari. Setelah penambahan pola transit, noise Gaussian disimulasikan untuk mencerminkan gangguan atau ketidakpastian yang umumnya ditemui dalam data pengamatan riil. Proses ini menghasilkan data sintetik yang realistis namun tetap terkendali untuk keperluan analisis algoritma.

B. Alur Implementasi

1. Mempersiapkan Library

Langkah pertama adalah mempersiapkan pustaka yang diperlukan untuk menjalankan program. Pustaka yang diperlukan termasuk NumPy, Matplotlib, Astropy,

dan TransitLeastSquares. Setiap pustaka memiliki peran spesifik dalam mendukung pembuatan data, implementasi algoritma, dan visualisasi hasil. NumPy digunakan untuk operasi numerik dan perhitungan matematis, sementara Matplotlib digunakan untuk visualisasi data seperti light curve dan grafik periodeogram. Astropy digunakan untuk implementasi algoritma BLS melalui modul BoxLeastSquares, yang dirancang khusus untuk mendeteksi pola transit berbentuk kotak. TransitLeastSquares digunakan untuk algoritma TLS yang mendeteksi transit berbasis model dengan sensitivitas tinggi terhadap sinyal eksoplanet.

Untuk memastikan kompatibilitas, pustaka-pustaka ini diinstal di awal menggunakan perintah pip install di Google Colab. Instalasi dilakukan di runtime Colab untuk memastikan bahwa semua pustaka memiliki versi terbaru yang mendukung implementasi algoritma. Contohnya, perintah !pip install transitleastsquares --quiet digunakan untuk menginstal pustaka TLS, sementara !pip install --upgrade astropy --quiet memastikan versi terbaru Astropy tersedia, termasuk modul BoxLeastSquares untuk analisis BLS. Dengan langkah ini, program siap untuk melanjutkan proses pembuatan data dummy yang akan digunakan sebagai input untuk kedua algoritma, memastikan kelancaran tanpa masalah kompatibilitas atau dependensi.

Berikut adalah *source code* untuk proses mempersiapkan library untuk program:

```
!pip install transitleastsquares --quiet
!pip install lightkurve --quiet
import numpy as np
import matplotlib.pyplot as plt
import time

# Algoritma BLS
from astropy.timeseries import BoxLeastSquares

# Algoritma TLS
from transitleastsquares import transitleastsquares, cleaned_array
```

Gambar 3.1 Proses Persiapan Library
sumber: https://github.com/Starath/MakalahMatdis

2. *Pembuatan Data Dummy*

Langkah selanjutnya dalam implementasi adalah menciptakan data dummy yang menyerupai kurva cahaya bintang dengan transit eksoplanet. Data ini dibuat dengan serangkaian waktu antara 1.000 hingga 10.000 titik, di mana setiap titik memiliki flux awal sebesar 1.0 untuk mencerminkan kecerlangan bintang yang stabil. Pola transit disimulasikan dengan mengurangi flux sebesar 1% setiap 2,5 hari (periode transit) selama 0,05 hari (durasi transit). Untuk meningkatkan realisme, noise Gaussian ditambahkan ke data untuk menghasilkan gangguan atau ketidakpastian, memungkinkan penilaian sensitivitas algoritma terhadap data yang tidak sempurna. Data dummy ini selanjutnya digunakan sebagai input untuk algoritma BLS dan TLS pada tahap selanjutnya.

Berikut adalah *source code* untuk proses pembuatan data dummy untuk diolah BLS dan TLS:

```
def generate_synthetic_lightcurve(num_points=1000,
                                  transit_period=2.5,
                                  transit_depth=0.01,
                                  noise_level=0.001,
                                  random_seed=42):
    np.random.seed(random_seed)

    # Waktu dalam hari, kita asumsikan data diambil selama ~10 hari
    time = np.linspace(0, 10, num_points)

    # Flux baseline = 1.0
    flux = np.ones(num_points)

    # Sisipkan transit sederhana berbentuk "box"
    # Transit terjadi jika (time % transit_period) < transit_width
    transit_width = 0.05
    for i, t in enumerate(time):
        phase = (t % transit_period)
        if phase < transit_width:
            flux[i] -= transit_depth  # turunkan flux sebesar transit_depth

    # Tambahkan noise Gaussian
    flux += np.random.normal(0, noise_level, num_points)

    return time, flux

```

Gambar 3.2 Proses Pembuatan Data Dummy
sumber: https://github.com/Starath/MakalahMatdis

3. *Penerapan Algoritma BLS*

Setelah pembuatan data dummy selesai, langkah berikutnya adalah menerapkan algoritma Box Least Squares (BLS) untuk mendeteksi pola transit pada kurva cahaya. Algoritma BLS didesain khusus untuk mendeteksi sinyal berbentuk kotak, yang sangat efektif dalam mengenali transit eksoplanet dengan penurunan flux yang tajam dan durasi yang singkat. Implementasi BLS dapat dilakukan dengan menggunakan modul BoxLeastSquares dari pustaka Astropy, yang menyediakan fungsi bawaan untuk melakukan pencarian transit dalam rentang periode tertentu.

Proses dimulai dengan menginisialisasi objek BLS menggunakan data waktu (time), flux, dan kesalahan pengukuran flux (flux_err). Biasanya, flux error diasumsikan kecil, misalnya 0,0001, untuk mensimulasikan tingkat ketidakpastian pengamatan. Selanjutnya, algoritma dijalankan pada rentang periode yang telah ditentukan, misalnya dari 0,5 hingga 5 hari, dengan pembagian periode menjadi 1.000 titik grid untuk presisi yang baik. Metode model.power() digunakan untuk menghitung kekuatan sinyal (power) pada setiap periode dalam grid, menghasilkan grafik periodeogram yang menunjukkan puncak pada periode di mana pola transit paling mungkin terjadi. Hasil ini divisualisasikan untuk membantu identifikasi periode transit utama, yang kemudian dapat dibandingkan dengan parameter data dummy untuk validasi.

Berikut adalah *source code* untuk proses penerapan algoritma BLS :

```
def run_bls(time, flux, period_min=0.5, period_max=5.0):
    # array "flux_err" untuk BLS
    flux_err = np.ones_like(flux) * 0.0001

    # Inisialisasi objek BLS
    model = BoxLeastSquares(time, flux, flux_err)

    # Menenentukan rentang periode (1000 titik periodik)
    periods = np.linspace(period_min, period_max, 1000)

    # Jalankan BLS (durasi transit diasumsikan tetap 0.02 hari untuk contoh)
    result = model.power(periods, 0.02)
    return result

```

Gambar 3.3 Algoritma BLS
sumber: https://github.com/Starath/MakalahMatdis

4. *Penerapan Algoritma TLS*

Setelah menerapkan algoritma BLS, langkah selanjutnya adalah menerapkan algoritma Transit Least Squares (TLS) untuk mengidentifikasi pola transit pada light curve. TLS dirancang untuk mengidentifikasi sinyal transit dengan sensitivitas yang lebih tinggi daripada BLS karena memanfaatkan model transit yang lebih realistis, termasuk bentuk kurva V atau U yang mencerminkan pola transit eksoplanet. Implementasi TLS dilakukan menggunakan pustaka TransitLeastSquares, yang menyediakan metode bawaan untuk analisis transit berdasarkan data waktu dan flux.

Proses dimulai dengan membersihkan data menggunakan fungsi cleaned_array untuk memastikan tidak ada nilai yang hilang (NaN) atau tidak valid. Selanjutnya, objek TLS diinisialisasi dengan data waktu (time) dan flux yang telah dibersihkan. Algoritma kemudian dijalankan menggunakan metode model.power(), yang melakukan pencarian transit dalam rentang periode tertentu berdasarkan grid otomatis yang disesuaikan oleh pustaka TLS. Hasil analisis ini menghasilkan grafik periodeogram yang menampilkan kekuatan sinyal transit pada setiap periode yang diuji, dengan puncak tertinggi (peak) menunjukkan periode transit yang paling mungkin.

Hasil TLS biasanya memberikan informasi tambahan, seperti kedalaman transit (transit depth), durasi transit, dan model kurva transit yang sesuai dengan data. Hal ini memungkinkan TLS untuk memberikan hasil yang lebih akurat dibandingkan BLS, terutama pada data dengan noise tinggi atau kedalaman transit yang sangat kecil. Sama halnya dengan BLS, hasil TLS ini divisualisasikan untuk membandingkan puncak pada periodeogram dengan parameter transit sebenarnya yang digunakan pada data dummy, sehingga validasi dapat dilakukan secara langsung.

Berikut adalah source code untuk proses penerapan algoritma TLS :

```python
def run_tls(time, flux):
    # Membersihkan data (misal jika ada NaN)
    time_clean, flux_clean = cleaned_array(time, flux)

    # Inisialisasi model TLS
    model = transitleastsquares(time_clean, flux_clean)

    # Jalankan TLS (dengan semua parameter default)
    results = model.power()
    return results

```

Gambar 3.4 Algoritma TLS
sumber: https://github.com/Starath/MakalahMatdis

5. *Analisis Kompleksitas Waktu*

Setelah menerapkan algoritma BLS dan TLS pada data dummy, langkah terakhirnya adalah menganalisis kompleksitas waktu eksekusi kedua algoritma. Analisis ini membandingkan efisiensi kedua algoritma dalam hal waktu proses terhadap dataset dengan ukuran yang berbeda. Pengukuran waktu eksekusi dilakukan menggunakan fungsi bawaan Python, yaitu time.perf_counter(), untuk merekam durasi proses dari awal hingga akhir setiap algoritma. Proses ini dibagi menjadi tiga bagian, yaitu penghitungan waktu eksekusi, pengujian pada berbagai ukuran dataset, dan visualisasi perbandingan hasil.

Pada bagian pertama, waktu eksekusi algoritma BLS dan TLS dihitung menggunakan fungsi compare_time_complexity. Fungsi ini menerima daftar ukuran dataset (n_sizes), seperti 500 hingga 8.000 titik data, lalu menghasilkan data dummy untuk setiap ukuran. Waktu eksekusi dihitung menggunakan time.perf_counter() dengan mencatat durasi proses untuk algoritma BLS dan TLS secara terpisah. Hasil pengukuran waktu disimpan dalam dua daftar, times_bls dan times_tls, untuk dianalisis lebih lanjut.

Berikut adalah *source code* untuk bagian perhitungan waktu eksekusi:

```python
def compare_time_complexity(n_sizes):
    times_bls = []
    times_tls = []

    for n in n_sizes:
        # Generate data dummy
        time, flux = generate_synthetic_lightcurve(num_points=n)

        # Timing BLS
        start_bls = time_module()
        _ = run_bls(time, flux)
        end_bls = time_module()
        times_bls.append(end_bls - start_bls)

        # Timing TLS
        start_tls = time_module()
        _ = run_tls(time, flux)
        end_tls = time_module()
        times_tls.append(end_tls - start_tls)

    return times_bls, times_tls

def time_module():
    return time.perf_counter()

```

Gambar 3.5 Perhitungan Waktu Eksekusi
sumber: https://github.com/Starath/MakalahMatdis

Pada bagian kedua, fungsi compare_time_complexity dipanggil untuk menguji algoritma pada berbagai ukuran dataset (n_sizes). Waktu eksekusi algoritma BLS dan TLS dihitung untuk setiap ukuran dataset, dan hasilnya disimpan dalam daftar times_bls dan times_tls. Output tersebut kemudian ditampilkan di layar untuk memastikan proses berjalan dengan benar, memuat daftar ukuran dataset beserta waktu eksekusi masing-masing algoritma.

Berikut adalah *source code* untuk bagian pengujian pada berbagai ukuran dataset:

```python
# Menguji dengan beberapa ukuran data yang berbeda
n_sizes = [500, 1000, 2000, 4000, 8000]

# Panggil fungsi pembanding waktu
times_bls, times_tls = compare_time_complexity(n_sizes)

# Tampilkan hasil di layar
print("Ukuran data (N):", n_sizes)
print("Waktu BLS (s):   ", times_bls)
print("Waktu TLS (s):   ", times_tls)
```

Gambar 3.6 Pengujian Dataset
sumber: https://github.com/Starath/MakalahMatdis

Algoritma tersebut menghasilkan output berikut:

```
Transit Least Squares TLS 1.32 (5 Apr 2024)
Creating model cache for 31 durations
Searching 499 data points, 713 periods from
0.602 to 4.99 days
Using all 2 CPU threads
100%|██████████| 713/713 periods |
00:06<00:00
Searching for best T0 for period 2.50078 days
Transit Least Squares TLS 1.32 (5 Apr 2024)
Creating model cache for 31 durations
Searching 999 data points, 714 periods from
0.602 to 4.995 days
Using all 2 CPU threads
100%|██████████| 714/714 periods |
00:09<00:00
Searching for best T0 for period 2.50433 days
Transit Least Squares TLS 1.32 (5 Apr 2024)
Creating model cache for 31 durations
Searching 1999 data points, 715 periods from
0.601 to 4.997 days
Using all 2 CPU threads
100%|██████████| 715/715 periods |
00:07<00:00
Searching for best T0 for period 2.49754 days
Transit Least Squares TLS 1.32 (5 Apr 2024)
Creating model cache for 31 durations
Searching 3999 data points, 715 periods from
0.601 to 4.999 days
Using all 2 CPU threads
100%|██████████| 715/715 periods |
00:12<00:00
Searching for best T0 for period 2.49842 days
Transit Least Squares TLS 1.32 (5 Apr 2024)
Creating model cache for 31 durations
Searching 7999 data points, 715 periods from
0.601 to 4.999 days
Using all 2 CPU threads
100%|██████████| 715/715 periods |
00:15<00:00
Searching for best T0 for period 2.49886 days
Ukuran data (N): [500, 1000, 2000, 4000,
8000]
Waktu BLS (s):    [0.025312980000080643,
0.027359697000065353, 0.04358363599999393,
0.057830918000036036, 0.09643961799997669]
Waktu TLS (s):    [6.5951527900001565,
9.539705378999997, 7.528058421999958,
12.744148288000133, 16.874778006000042]
```

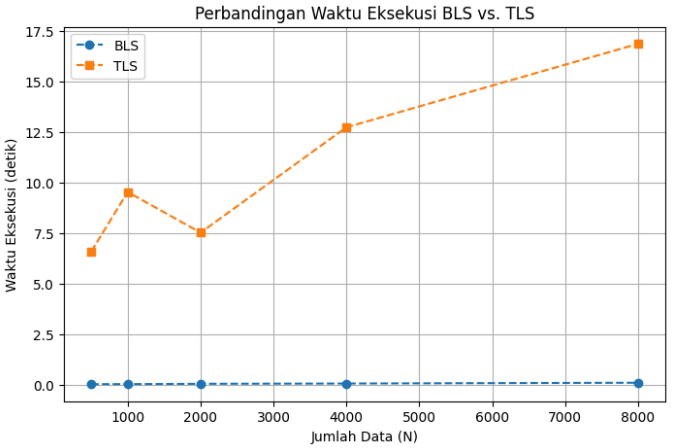
Proses ini diakhiri dengan bagian visualisasi hasil eksekusi waktu dari BLS dan TLS dalam bentuk grafik. Grafik tersebut menampilkan ukuran dataset pada sumbu X dan waktu eksekusi pada sumbu Y, dengan dua kurva terpisah yang mewakili masing-masing algoritma. Dengan visualisasi ini, analisis efisiensi kedua algoritma menjadi lebih mudah, terutama dalam menyoroti perbedaan kinerja saat ukuran dataset bertambah. Grafik ini membantu dalam memahami skalabilitas masing-masing algoritma secara langsung.

Berikut adalah *source code* untuk bagian visualisasi hasil eksekusi waktu:

```python
plt.figure(figsize=(8,5))
plt.plot(n_sizes, times_bls, 'o--', label='BLS')
plt.plot(n_sizes, times_tls, 's--', label='TLS')
plt.xlabel('Jumlah Data (N)')
plt.ylabel('Waktu Eksekusi (detik)')
plt.title('Perbandingan Waktu Eksekusi BLS vs. TLS')
plt.legend()
plt.grid(True)
plt.show()
```

Gambar 3.7 Proses Visualisasi Eksekusi Waktu
sumber: https://github.com/Starath/MakalahMatdis

Algoritma tersebut menghasilkan output berikut:



Gambar 3.8 Visualisasi Perbandingan Waktu Eksekusi
sumber: https://github.com/Starath/MakalahMatdis

IV. Pembahasan

A. Hasil Implementasi

Hasil implementasi menunjukkan bahwa algoritma BLS secara signifikan lebih cepat dibandingkan TLS untuk semua ukuran dataset yang diuji. Waktu eksekusi BLS berkisar antara 0,025 detik hingga 0,096 detik, dengan peningkatan waktu yang hampir linear seiring bertambahnya jumlah data. Sebaliknya, TLS membutuhkan waktu antara 6,59 detik hingga 16,87 detik, dengan kenaikan yang lebih tajam pada dataset yang lebih besar. Perbedaan ini disebabkan oleh pendekatan TLS yang lebih kompleks, yang mencakup pemodelan transit berbentuk kurva U/V dan evaluasi durasi transit yang beragam, sementara BLS hanya menggunakan model transit sederhana berbentuk box.

Meskipun waktu eksekusi TLS lebih tinggi, algoritma ini

unggul dalam mendeteksi transit dengan sinyal lemah atau durasi pendek berkat pemodelannya yang lebih realistis. Kedua algoritma berhasil mengidentifikasi periode transit sekitar 2,5 hari, yang sesuai dengan data dummy, tetapi TLS memberikan hasil yang lebih mendetail. Dengan skalabilitas yang lebih baik, BLS lebih cocok untuk dataset besar dengan kebutuhan analisis cepat, sedangkan TLS ideal untuk studi mendalam yang membutuhkan akurasi tinggi. Diagram hasil mendukung kesimpulan ini, menampilkan efisiensi BLS yang lebih stabil dibandingkan TLS.

A. Analisis Kompleksitas

Analisis kompleksitas waktu menunjukkan perbedaan signifikan antara algoritma BLS dan TLS dalam hal efisiensi komputasi. Algoritma BLS memiliki kompleksitas waktu O(N × M), di mana N adalah jumlah titik data dan M adalah jumlah kandidat periode yang diuji. Dalam implementasi ini, BLS mengevaluasi setiap kandidat periode dengan pendekatan box-fitting yang sederhana, sehingga waktu eksekusinya meningkat secara linier dengan jumlah titik data. Hal ini tercermin pada hasil yang menunjukkan waktu eksekusi BLS tetap rendah, bahkan ketika ukuran dataset bertambah, misalnya dari 0,025 detik pada 500 titik data menjadi hanya 0,096 detik pada 8.000 titik data.

Di sisi lain, algoritma TLS memiliki kompleksitas waktu yang lebih tinggi, yaitu O(P × D × N), di mana P adalah jumlah kandidat periode, D adalah jumlah durasi transit yang diuji, dan N adalah jumlah titik data. TLS melakukan evaluasi terhadap kombinasi periode dan durasi transit dengan memodelkan kurva transit yang lebih realistis, seperti bentuk U/V. Hal ini menyebabkan waktu eksekusi TLS meningkat signifikan seiring bertambahnya ukuran dataset, seperti terlihat pada hasil, dari 6,59 detik pada 500 titik data menjadi 16,87 detik pada 8.000 titik data. Pendekatan kompleks ini memberikan keunggulan TLS dalam mendeteksi transit dengan sinyal lemah, tetapi dengan biaya komputasi yang jauh lebih besar dibandingkan BLS.

Perbandingan kompleksitas waktu ini menunjukkan bahwa BLS lebih efisien untuk dataset besar dengan kebutuhan analisis cepat, sementara TLS lebih sesuai untuk analisis mendalam yang memprioritaskan akurasi tinggi pada sinyal transit yang lebih kompleks. Diagram waktu eksekusi juga mendukung analisis ini, dengan kurva waktu BLS yang lebih datar dibandingkan TLS. Hal ini menjadikan BLS lebih skalabel untuk dataset astronomi berskala besar, sementara TLS dapat dioptimalkan dengan pendekatan seperti paralelisasi untuk mengurangi waktu eksekusinya.

V. Kesimpulan

Dari implementasi dan pembahasan tersebut, dapat disimpulkan bahwa algoritma Box Least Squares (BLS) dan Transit Least Squares (TLS) memiliki kelebihan dan kekurangan dalam deteksi transit eksoplanet. BLS menunjukkan efisiensi tinggi dan waktu eksekusi yang cepat, terutama pada dataset besar. Kompleksitas waktu algoritma BLS adalah O(N × M), di mana N adalah jumlah titik data dan M adalah jumlah kandidat periode. Dengan pendekatan fitting-kotak yang sederhana, BLS cocok untuk analisis cepat, terutama dalam penyaringan awal data light curve dalam jumlah besar. Namun, pendekatan tersebut cenderung kurang akurat dalam mendeteksi transit dengan sinyal lemah atau bentuk transit yang tidak ideal.

Sementara itu, TLS menawarkan sensitivitas yang lebih tinggi dengan menggambarkan transit sebagai kurva U/V yang lebih realistis, namun memerlukan waktu eksekusi yang lebih lama. Kompleksitas waktu TLS adalah O(P × D × N), di mana P adalah jumlah kandidat periode, D adalah jumlah durasi transit yang diuji, dan N adalah jumlah titik data. Hal ini menjadikan TLS lebih sesuai untuk analisis mendalam pada dataset dengan sinyal transit yang sulit dideteksi. Data perbandingan waktu eksekusi menunjukkan bahwa TLS memerlukan waktu lebih lama seiring dengan meningkatnya ukuran dataset. Oleh karena itu, pemilihan algoritma sebaiknya didasarkan pada kebutuhan analisis yang spesifik, apakah itu memprioritaskan kecepatan pemrosesan atau akurasi deteksi transit.

VI. Lampiran

full source code ada di dalam repository berikut:
https://github.com/Starath/MakalahMatdis.git

VII. Ucapan Terima Kasih

Penulis ingin mengucapkan terima kasih yang sebesar-besarnya kepada Tuhan Yang Maha Esa yang telah membantu mewujudkan makalah ini. Penulis juga mengucapkan terima kasih kepada dosen pengampu mata kuliah Matematika Diskrit, Ir. Rila Mandala, M.Eng., Ph.D., yang telah berdedikasi dalam mengajar mata kuliah ini dan membantu penulis dalam memahami dasar-dasar Matematika Diskrit sehingga sangat membantu penulis dalam mengaplikasikannya ke dalam makalah ini.

References

[1] Hippke, M., & Heller, R. (2019). Transit least-squares survey – I. Discovery and validation of the exoplanet host star TOI-251. *Astronomy & Astrophysics*, *623*, A39. https://doi.org/10.1051/0004-6361/201834672. [Diakses 8 January 2025].

[2] Kovács, G., Zucker, S., & Mazeh, T. (2002). A box-fitting algorithm in the search for periodic transits. *Astronomy & Astrophysics*, *391*(1), 369–377. https://doi.org/10.1051/0004-6361:20020802. [Diakses 8 January 2025].

[3] Seager, S., & Mallén-Ornelas, G. (2003). A unique solution of planetary transit light curves: Light curve analysis for eccentric orbits. The Astrophysical Journal, 585(2), 1038–1055. https://doi.org/10.1086/346105. [Diakses 8 January 2025].

[4] Munir, Rinaldi. 2023. "Kompleksitas Algoritma (Bagian 1)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/25-Kompleksitas-Algoritma-Bagian1-2024.pdf. [Diakses 8 January 2025].

[5] Munir, Rinaldi. 2023. "Kompleksitas Algoritma (Bagian 2)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/26-Kompleksitas-Algoritma-Bagian2-2024.pdf. [Diakses 8th January 2025].

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, 26 Desember 2024

Athian Nugraha Muarjuang
13523106