Prosiding Seminar Implementasi Teknologi Informasi dan Komunikasi
Vol. 4, No. 1, Januari 2025
ISSN: 2809-9834, DOI: 10.31284/p.semtik.2025-1.7008

Penyelesaian Traveling Salesman Problem Dengan Algoritma Ant Colony Menggunakan Multi Processing dan Multi Threading Parallel Programming

Muhammad Andika Saputra, Abdul Rahim, Mufti Kholil Romadhoni*, Muhammad Shochibul Burhan

Program Studi Teknik Informatika, Universitas Muhammadiyah Malang

*Penulis korespondensi. E-mail: muftikholil28@gmail.com

ABSTRACT

Traveling Salesman Problem (TSP) is a key challenge in combinatorial optimization, where finding the shortest route becomes difficult as the scale of the route increases. Ant Colony Optimization (ACO) is a popular method based on ant behavior to explore optimal solutions. However, ACO often requires high computation time, especially for large-scale problems. This research proposes the incorporation of parallel programming through multi-processing and multi-threading to speed up the execution without compromising the solution quality. The implementation results show that multithreading provides the fastest execution time with consistent fitness value, which is 0.0030 at iteration 100 and 0.0022-0.0024 at iteration 1000. At iteration 100, multithreading recorded 0.0023-0.0037 seconds, while multiprocessing required 0.1889-0.2041 seconds. At 1000 iterations, multithreading reached 0.0032-0.0042 seconds, while multiprocessing ranged from 0.1919-0.2266 seconds. Multithreading improves the efficiency of execution time to about 99.83% compared to standard ACO at iteration 100, and about 99.80% at iteration 1000, without compromising the solution quality. This study confirms that multithreading is more efficient than multiprocessing for solving TSP using ACO.

Keywords

Ant Colony, Multi Processing, Multi Threading, Parallel programming, Traveling Salesman Problem

ABSTRAK

Traveling Salesman Problem (TSP) adalah tantangan utama dalam optimasi kombinatorial, di mana menemukan rute terpendek menjadi sulit dengan bertambahnya skala rute. Ant Colony Optimization (ACO) adalah metode populer berbasis perilaku semut untuk menjelajahi solusi optimal. Namun, ACO sering membutuhkan waktu komputasi tinggi, terutama untuk masalah berskala besar. Penelitian ini mengusulkan penggabungan pemrograman paralel melalui multiprocessing dan multithreading untuk mempercepat eksekusi tanpa mengurangi kualitas solusi. Hasil implementasi menunjukkan bahwa multithreading memberikan waktu eksekusi tercepat dengan nilai fitness konsisten, yaitu 0.0030 pada iterasi 100 dan 0.0022–0.0024 pada iterasi 1000. Pada iterasi 100, multithreading mencatat waktu 0.0023–0.0037 detik, sementara multiprocessing membutuhkan 0.1889–0.2041 detik. Pada iterasi 1000, multithreading mencapai 0.0032–0.0042 detik, sedangkan multiprocessing berkisar antara 0.1919–0.2266 detik. Multithreading meningkatkan efisiensi waktu eksekusi hingga sekitar 99,83% dibandingkan dengan ACO standar pada iterasi 100, dan sekitar 99,80% pada iterasi 1000, tanpa mengorbankan kualitas solusi. Penelitian ini menegaskan bahwa multithreading lebih efisien dibandingkan multiprocessing untuk menyelesaikan TSP menggunakan ACO.

PENDAHULUAN

Traveling Salesman Problem (TSP) merupakan salah satu masalah optimasi kombinatorial yang paling dikenal dan sering dipelajari dalam ilmu komputer dan riset operasi. TSP berusaha mencari rute terpendek untuk mencapai suatu node, di mana seorang salesman harus mengunjungi setiap node sekali saja dan kembali ke node asal dengan jarak tempuh total yang paling pendek. Tujuannya adalah untuk menemukan rute perjalanan dengan total jarak tempuh minimal [1]. Meskipun formulasi TSP tampak sederhana, masalah ini termasuk dalam kategori NP-hard, yang berarti bahwa ketika jumlah node bertambah, kompleksitas untuk menemukan solusi optimal meningkat secara eksponensial. Hal ini membuat TSP sangat sulit diselesaikan secara efisien menggunakan algoritma deterministik biasa, terutama pada skala besar [2][3], [4], [5].lam implementasinya, TSP memiliki banyak aplikasi yang telah dikembangkan untuk mengatasi kompleksitas yang dapat memberikan solusi yang cukup baik meski tidak selalu optimal. Pendekatan-pendekatan ini telah terbukti efektif dalam menangani kasus TSP pada dunia nyata dengan jumlah node yang besar [6], [7], [8], [9]. Oleh karena itu, berbagai pendekatan heuristik dan

metaheuristik telah dikembangkan untuk menghasilkan solusi yang mendekati optimal dengan waktu komputasi yang dapat diterima. Salah satu pendekatan yang populer adalah Ant Colony Optimization (ACO), yang telah terbukti efektif dalam menangani masalah TSP skala besar [7], [10], [11] .Penelitian terbaru juga menunjukkan peningkatan dalam algoritma ACO yang dirancang khusus untuk TSP, seperti pengembangan algoritma ant colony berbasis peningkatan (Improved Ant Colony-Based Algorithm) yang terbukti lebih efisien dalam menyelesaikan permasalahan ini [12]. Ant Colony Optimization (ACO) diadopsi dari perilaku alami koloni semut yang mampu menemukan rute terpendek dalam perjalanan dari sarang menuju sumber makanan dan kembali lagi. Selama perjalanan menuju sumber makanan, semut meninggalkan jejak berupa zat pada setiap tempat yang dilalui guna menandai rute tersebut. Zat yang digunakan disebut pheromone, yang berfungsi sebagai alat komunikasi antar semut dalam membangun rute. Dalam algoritma ACO, agen-agen buatan bertindak seperti semut yang menjelajahi ruang solusi masalah dan secara bertahap membangun solusi yang lebih baik dengan memanfaatkan jejak pheromone yang diperbarui berdasarkan kualitas solusi yang ditemukan pada iterasi sebelumnya [13].

Ant Colony Optimization (ACO) telah diaplikasikan dalam berbagai konteks, seperti proses pengumpulan sampah, untuk menemukan jalur terpendek yang optimal. Hal ini menunjukkan fleksibilitas ACO dalam menyelesaikan permasalahan rute dengan efisien dan memberikan contoh penerapan ACO di dunia nyata, yang relevan dengan permasalahan TSP [14].

Seiring dengan bertambahnya ukuran masalah, seperti jumlah node yang harus dikunjungi dalam TSP, kebutuhan komputasi untuk menjalankan algoritma ACO secara signifikan meningkat [13]. Hal ini disebabkan karena setiap semut dalam koloni harus melakukan perhitungan probabilitas untuk setiap perpindahan antar node, dan proses ini harus diulang untuk beberapa iterasi untuk mencapai konvergensi. Selain itu, penyesuaian nilai pheromone pada setiap jalur juga memerlukan waktu komputasi yang tidak sedikit, terutama ketika jumlah node sangat besar [15], [16] .Penelitian sebelumnya menunjukkan bahwa ACO memiliki beberapa kekurangan, termasuk waktu komputasi yang lambat dan masalah stagnasi, di mana semua semut cenderung fokus pada satu jalur dengan konsentrasi pheromone tertinggi[17], [18]. Peningkatan efisiensi dalam algoritma ACO melalui pendekatan parallel juga telah terbukti mampu mengatasi masalah ini, terutama dalam menangani TSP berskala besar [3].

Dalam beberapa tahun terakhir, komputasi paralel telah meningkatkan efisiensi algoritma metaheuristik seperti ACO. Dengan komputasi paralel, ACO dapat berjalan simultan di beberapa prosesor atau mesin, memungkinkan eksplorasi lebih banyak solusi secara bersamaan. Teknik ini mendistribusikan beban komputasi secara merata, mengurangi waktu untuk mencapai konvergensi menuju solusi optimal [19], [20], [21]. Dengan multi-processing atau parallel programming, eksplorasi TSP menjadi lebih cepat, meningkatkan efisiensi komputasi [2], [3], [13], [4].

Penelitian ini berfokus pada penerapan multi-processing dan multi-threading dalam ACO untuk menyelesaikan TSP, bertujuan mempercepat pencarian solusi berkualitas untuk masalah berskala besar [4], [18]. Analisis mencakup dampak paralelisasi terhadap kecepatan konvergensi dan kualitas solusi, dengan evaluasi berbasis kecepatan dan hasil eksperimen.

Dalam pengelolaan limbah, penelitian oleh Ismail, Krisnaputra, dan Bahiuddin menerapkan ACO untuk rute terpendek, mengatasi konsumsi energi berlebih, waktu yang tidak efisien, dan emisi polusi. ACO menghasilkan rute 752 meter, mengurangi bahan bakar fosil dan meningkatkan efisiensi operasional [14]. Pendekatan ini mendukung kota pintar dan teknologi IoT, memberikan solusi inovatif untuk pengelolaan limbah yang ramah lingkungan dan berkelanjutan.

TINJAUAN PUSTAKA

Traveling Salesman Problem (TSP)

Traveling Salesman Problem (TSP) merupakan salah satu masalah optimasi kombinatorial yang paling dikenal dalam riset operasi dan ilmu komputer. TSP bertujuan untuk menemukan rute terpendek bagi seorang salesman yang harus mengunjungi setiap node tepat satu kali dan kembali ke node awal dengan jarak total minimal [1]. Meskipun formulasi TSP tampak sederhana, permasalahan ini termasuk dalam kategori NP-hard, yang berarti kompleksitasnya meningkat secara eksponensial

seiring bertambahnya jumlah node. Oleh karena itu, pendekatan heuristik dan metaheuristik seperti Ant Colony Optimization (ACO) telah banyak diterapkan untuk mengatasi tantangan ini [2][3].

Ant Colony Optimization (ACO)

Ant Colony Optimization (ACO) merupakan algoritma metaheuristik yang diilhami oleh perilaku alami koloni semut dalam mencari rute terpendek dari sarang menuju sumber makanan. ACO menggunakan agen buatan ("semut") untuk menjelajahi ruang solusi dan memperbarui rute berdasarkan jejak feromon yang ditinggalkan [4]. Jejak feromon yang lebih kuat menunjukkan kualitas solusi yang lebih baik, sehingga probabilitas pemilihan jalur tersebut meningkat. Dalam implementasi pada TSP, ACO telah terbukti efektif dalam menghasilkan solusi mendekati optimal, meskipun memerlukan waktu komputasi tinggi, terutama untuk masalah berskala besar [5][6].

Pendekatan Paralel dalam ACO

Dalam beberapa tahun terakhir, penerapan komputasi paralel seperti multi-threading dan multi-processing menjadi solusi untuk meningkatkan efisiensi ACO. Multi-threading memungkinkan distribusi tugas antar thread dalam satu prosesor, sementara multi-processing membagi tugas antar beberapa prosesor untuk mengurangi waktu eksekusi. Penelitian sebelumnya menunjukkan bahwa pendekatan paralel dapat mempercepat konvergensi solusi tanpa mengorbankan kualitasnya [7][8].

METODE

Penelitian Traveling Salesman Problem (TSP) ini mengimplementasikan metode Ant Colony Optimization (ACO) yang terintegrasi dengan pemrograman paralel melalui pendekatan multi-threading dan multi-processing. Penerapan integrasi antar kedua metode ini bertujuan untuk mempercepat waktu komputasi dan meningkatkan kualitas solusi melalui eksekusi paralel pada beberapa prosesor. Algoritma ACO, yang terinspirasi dari perilaku semut dalam menemukan jalur terpendek menuju sumber makanan, digunakan sebagai algoritma utama untuk mencari solusi optimal dari Traveling Salesman Problem (TSP). Untuk mengoptimalkan efisiensi dan waktu komputasi, algoritma ACO diimplementasikan dalam tiga variasi: ACO biasa, ACO dengan multi-threading, dan ACO dengan multi-processing [22].

Setiap variasi algoritma diujicobakan tiga kali untuk memastikan konsistensi hasil, dengan 50 agen semut, 5 solusi terbaik diambil setiap iterasi, dan faktor peluruhan feromon 0,95. Pengujian dilakukan dalam 100 dan 1000 iterasi untuk menilai performa algoritma dalam mencari solusi optimal.Tahapan penelitian meliputi beberapa langkah. Pertama, Inisialisasi Parameter ACO dilakukan dengan menetapkan jumlah agen semut, solusi terbaik, faktor peluruhan feromon, dan jumlah iterasi. Kedua, Inisialisasi Multithreading dan Multiprocessing mencakup penggunaan empat thread untuk multi-threading dan empat prosesor untuk multi-processing. Ketiga, Pencarian Jalur Optimal dilakukan melalui tiga pendekatan: ACO konvensional secara seri, ACO multi-threading dengan tugas didistribusikan ke thread, dan ACO multi-processing dengan pembagian tugas pada prosesor. Terakhir, Evaluasi dan Analisis Hasil mencakup penghitungan waktu eksekusi dan kualitas solusi sebagai fitness, dengan pengujian setiap metode dilakukan tiga kali untuk mendapatkan rata-rata performa.

Data Jarak Antar Node

Data berasal dari sebuah PT.XYZ yang menggunakan informasi jarak antar kota untuk keperluan logistik dan distribusi. Dalam tabel ini, setiap elemen merepresentasikan jarak antara dua kota tertentu dalam satuan kilometer (km). Data ini disusun dalam bentuk matriks simetris, di mana nilai di persimpangan baris dan kolom menunjukkan jarak langsung antara dua kota tersebut. Sebagai contoh, jarak antara Kota 1 dan Kota 2 adalah 29 km, sementara jarak antara Kota 1 dan Kota 3 adalah 20 km. Perusahaan menggunakan data ini untuk merencanakan rute distribusi barang secara optimal, dengan tujuan mengurangi waktu tempuh dan biaya transportasi. Dengan memanfaatkan data ini, perusahaan dapat menentukan rute tercepat atau terpendek yang perlu ditempuh oleh armada

distribusi saat mengirimkan produk dari satu kota ke kota lain. Ini tidak hanya meningkatkan efisiensi distribusi, tetapi juga membantu perusahaan menghemat sumber daya, seperti bahan bakar dan tenaga kerja, serta meningkatkan ketepatan waktu pengiriman kepada konsumen.

Perhitungan Algoritma Ant Colony

Algoritma Ant Colony Optimization (ACO) adalah algoritma berbasis metaheuristik yang meniru perilaku semut dalam mencari jalur terpendek dari sarang ke sumber makanan. Algoritma ini sangat cocok untuk masalah optimasi kombinatorial seperti Traveling Salesman Problem (TSP), di mana tujuan utamanya adalah menemukan rute terpendek yang mengunjungi setiap kota tepat sekali dan kembali ke kota awal [23]. Dalam ACO, sejumlah agen (semut) melakukan pencarian solusi dengan mengikuti "feromon" yang disimpan di jalur-jalur yang dilalui. Jalur dengan konsentrasi feromon lebih tinggi cenderung dipilih oleh semut lainnya, sehingga meningkatkan probabilitas bahwa jalur tersebut akan digunakan kembali, dan secara bertahap menciptakan rute yang optimal.

Berikut adalah proses utama dalam Algoritma Ant Colony Optimization:

1. Inisialisasi Feromon: Pada awal algoritma, feromon di setiap jalur antar kota diinisialisasi dengan nilai yang sama, misalnya τ_0, dan akan diperbarui seiring berjalannya algoritma.
2. Pemilihan Jalur oleh Semut: Semut memilih jalur berdasarkan probabilitas yang dihitung dengan mempertimbangkan jumlah feromon dan jarak antar kota. Semakin besar feromon dan semakin pendek jarak, semakin besar probabilitas jalur tersebut.

$$P_{ij}^{(k)} = \frac{(\tau_{ij})^{\alpha} \cdot (\eta_{ij})^{\beta}}{\sum_{l \in \text{allowed}} (\tau_{il})^{\alpha} \cdot (\eta_{il})^{\beta}}$$

Gambar 1. Rumus probabilitas pemilihan jalur

3. Update Feromon: Setelah setiap semut menyelesaikan perjalanannya, feromon di jalur yang dilalui diperkuat, sementara feromon di jalur lainnya mengalami evaporasi. Rumus update feromon untuk setiap jalur (i) (j) adalah:

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{(k)}$$

Gambar 2. Rumus update feromon

4. Iterasi dan Konvergensi: Proses pemilihan jalur dan update feromon diulang hingga konvergensi tercapai, yaitu ketika rute yang ditemukan tidak berubah signifikan atau batas iterasi tercapai.

Metode Paralel dengan MultiThread

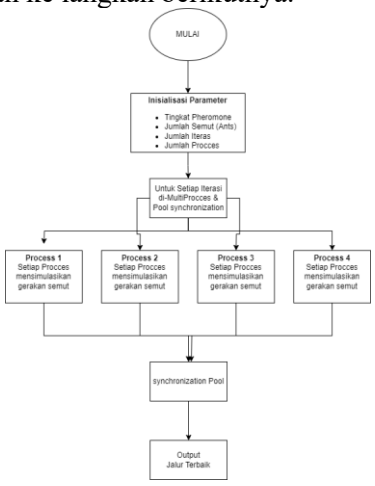
Metode parallel digunakan dalam algoritma Ant Colony Optimization (ACO) untuk membagi proses simulasi pergerakan semut menjadi beberapa thread, seperti yang ditunjukkan pada Gambar 3. Setiap thread menjalankan simulasi pergerakan semut secara bersamaan pada iterasi yang sama, dengan synchronization barrier di antara iterasi untuk memastikan data terbaru diperbarui secara serentak sebelum melanjutkan ke langkah berikutnya.

Gambar 3. Process parallel MultiThread ACO dengan metode synchronization barrier

Bagan tersebut menggambarkan penerapan algoritma Ant Colony Optimization (ACO) menggunakan pemrograman paralel berbasis multi-threading untuk mempercepat pencarian solusi optimal. Proses dimulai dengan inisialisasi parameter seperti tingkat pheromone, jumlah semut, iterasi, dan jumlah thread, yang menentukan pergerakan semut dalam menjelajahi ruang solusi. Eksplorasi solusi dilakukan secara paralel, di mana setiap thread mensimulasikan pergerakan satu semut secara independen berdasarkan tingkat pheromone dan informasi heuristik, sehingga memanfaatkan CPU secara optimal, mengurangi waktu komputasi, dan meningkatkan efisiensi tanpa mengurangi akurasi hasil. Setelah setiap thread menyelesaikan tugasnya, hasil dari seluruh thread disinkronkan melalui penghalang sinkronisasi untuk memperbarui pheromone secara serentak sebelum iterasi berikutnya, memastikan semua solusi diperhitungkan secara kolektif. Proses ini diulang hingga ditemukan solusi terbaik dari seluruh iterasi, yang mendekati atau merupakan solusi optimal untuk masalah seperti Traveling Salesman Problem. Implementasi kode dilakukan dengan berbagai percobaan menggunakan iterasi dan jumlah thread yang sesuai dengan jumlah semut, serta matriks jarak kompleks untuk menguji kinerja dalam skala besar, menunjukkan bahwa pendekatan multi-threading ini mampu mempercepat proses komputasi secara signifikan.

Metode Paralel dengan MultiProcessing

Metode parallel digunakan dalam algoritma Ant Colony Optimization (ACO) untuk membagi proses simulasi pergerakan semut menjadi beberapa thread, seperti yang ditunjukkan pada Gambar 4. Setiap thread menjalankan simulasi pergerakan semut secara bersamaan pada iterasi yang sama, dengan synchronization barrier di antara iterasi untuk memastikan data terbaru diperbarui secara serentak sebelum melanjutkan ke langkah berikutnya.



Gambar 4. Process parallel pada ACO dengan metode synchronization pool

Bagan tersebut menjelaskan alur penerapan algoritma Ant Colony Optimization (ACO) menggunakan multi-processing parallel programming untuk mempercepat pencarian solusi optimal. Proses diawali dengan inisialisasi parameter, seperti tingkat pheromone, jumlah semut, iterasi, dan jumlah prosesor, untuk mempersiapkan algoritma sebelum simulasi dimulai. Pada tahap pemrosesan paralel, tugas distribusi solusi dilakukan oleh empat prosesor utama, di mana setiap prosesor menjalankan simulasi pergerakan semut secara independen pada subset solusi yang berbeda, memanfaatkan multi-processing untuk meningkatkan efisiensi komputasi melalui distribusi tugas yang merata. Setelah setiap proses selesai dalam iterasi tertentu, hasil dikumpulkan dan disinkronkan melalui synchronization pool untuk mengevaluasi dan memperbaiki data berdasarkan solusi terbaik yang ditemukan. Pada akhir iterasi, jalur terbaik dari seluruh proses sinkronisasi dipilih sebagai output, merepresentasikan solusi optimal atau mendekati optimal untuk masalah seperti Traveling Salesman Problem (TSP). Implementasi algoritma dilakukan dengan tiga percobaan masing-masing 100 iterasi menggunakan 4 prosesor, yang menunjukkan bahwa pendekatan multi-processing ini mampu mempercepat komputasi pada matriks jarak besar dan kompleks tanpa mengurangi akurasi hasil.

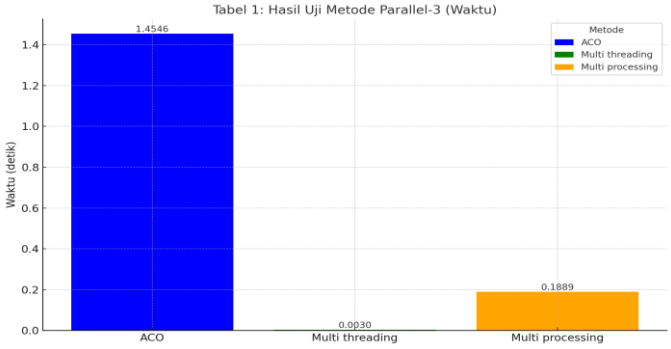
HASIL DAN PEMBAHASAN

Hasil pengujian dari implementasi Ant Colony Optimization (ACO) dengan metode multi-threading dan multi processing akan dibahas pada bagian ini. Tujuan dari pengujian adalah untuk melihat seberapa efektif dan efisien kedua metode paralel ini dalam menyelesaikan Traveling Salesman Problem (TSP), khususnya dari segi waktu eksekusi dan kualitas solusi yang diukur dengan fitness. Pengujian dilakukan dengan menggunakan iterasi 100 dan 1000, sehingga dapat terlihat bagaimana kinerja masing-masing metode pada skala yang berbeda. Hasil pengujian ini disajikan dalam bentuk tabel untuk memberikan gambaran yang lebih jelas terkait perbandingan waktu dan kualitas antar metode [25] .

Dari hasil uji coba menggunakan parallel programing didapatkan hasil yang ditunjukan pada Tabel 1 menggunakan iterasi 100 dan Tabel 2 menggunakan iterasi 1000.

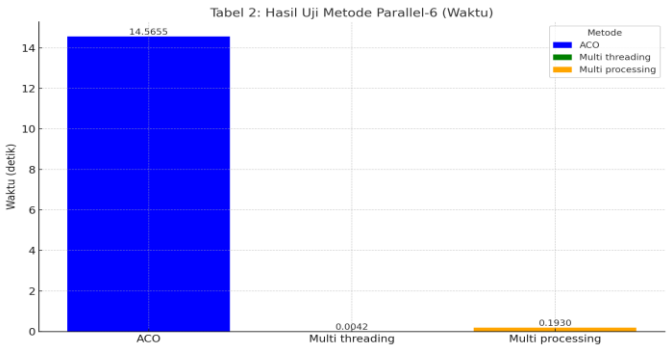
Tabel 1. Hasil uji metode parallel iterasi 100

Metode	Jml. Iterasi	Fitnes	Waktu
ACO	100	0.0030	1.4546 detik
Multi threading	100	0.0024	0.0030 detik
Multi processing	100	0.0024	0.1889 detik



Tabel 2. Hasil uji metode parallel -6

Metode	Jml. Iterasi	Fitnes	Waktu
ACO	1000	0.0030	14.5655 detik
Multi threading	1000	0.0022	0.0042 detik
Multi processing	1000	0.0024	0.1930 detik



KESIMPULAN

Hasil pengujian menunjukkan bahwa ACO dengan parallel computing menggunakan Multithreading dan Multiprocessing secara signifikan meningkatkan efisiensi dibandingkan ACO konvensional. ACO konvensional memiliki waktu eksekusi 1.4546–1.5265 detik pada 100 iterasi dan 14.5655–15.0815 detik pada 1000 iterasi. Multithreading mencatat waktu tercepat, yaitu 0.0023–0.0042 detik, sekitar 99,83%–99,94% lebih cepat pada iterasi 100 dan 99,80% pada iterasi 1000, sementara Multiprocessing membutuhkan 0.1889–0.2266 detik, lebih lambat dari Multithreading tetapi tetap jauh lebih cepat dibandingkan ACO konvensional. Paralelisasi juga menunjukkan stabilitas waktu eksekusi yang lebih konsisten. Dalam hal fitness, ACO konvensional menghasilkan nilai tetap 0.0030, sedangkan Multithreading mencapai 0.0022–0.0024 dan Multiprocessing 0.0024–0.0025, menunjukkan bahwa paralelisasi tidak hanya mempercepat eksekusi tetapi juga meningkatkan kualitas solusi. Multithreading disarankan untuk fokus pada kecepatan dan optimasi solusi, sementara Multiprocessing cocok untuk keseimbangan antara kecepatan dan penggunaan sumber daya. ACO konvensional hanya direkomendasikan jika paralelisasi tidak memungkinkan.

DAFTAR PUSTAKA

[1] Z. hong Jia, X. xue Zhuo, J. Y. T. Leung, and K. Li, “Integrated production and transportation on parallel batch machines to minimize total weighted delivery time,” Comput Oper Res, vol. 102, pp. 39–51, Feb. 2019, doi: 10.1016/j.cor.2018.07.026.

[2] L. Yang, T. Jiang, and R. Cheng, “Tensorized Ant Colony Optimization for GPU Acceleration,” 2024, doi: 10.1145/3638530.

[3] R. Skinderowicz, “Improving Ant Colony Optimization efficiency for solving large TSP instances,” Appl Soft Comput, vol. 120, May 2022, doi: 10.1016/j.asoc.2022.108653.

[4] D. M. Chitty, “Applying ACO To Large Scale TSP Instances,” Sep. 2017, doi: 10.1007/978-3-319-66939-7_9.

[5] S. Chowdhury, M. Marufuzzaman, H. Tunc, L. Bian, and W. Bullington, “A modified Ant Colony Optimization algorithm to solve a dynamic traveling salesman problem: A case study with drones for wildlife surveillance,” J Comput Des Eng, vol. 6, no. 3, pp. 368–386, Jul. 2019, doi: 10.1016/j.jcde.2018.10.004.

[6] J. M. Cecilia, A. Llanes, J. L. Abellán, J. Gómez-Luna, L. W. Chang, and W. M. W. Hwu, “High-throughput Ant Colony Optimization on graphics processing units,” J Parallel Distrib Comput, vol. 113, pp. 261–274, Mar. 2018, doi: 10.1016/j.jpdc.2017.12.002.

[7] R. Skinderowicz, “Implementing a GPU-based parallel MAX–MIN Ant System,” Future Generation Computer Systems, vol. 106, pp. 277–295, May 2020, doi: 10.1016/j.future.2020.01.011.

[8] M. Pedemonte, S. Nesmachnow, and H. Cancela, “A survey on parallel ant colony optimization,” Applied Soft Computing Journal, vol. 11, no. 8, pp. 5181–5197, 2011, doi: 10.1016/j.asoc.2011.05.042.

[9] Z. Wu, “A comparative study of solving traveling salesman problem with genetic algorithm, ant colony algorithm, and particle swarm optimization,” in ACM International Conference

Proceeding Series, Association for Computing Machinery, Dec. 2020, pp. 95–99. doi: 10.1145/3450292.3450308.

[10] A. M. Abdelmoaty and I. I. Ibrahim, "Comparative Analysis of Four Prominent Ant Colony Optimization Variants: Ant System, Rank-Based Ant System, Max-Min Ant System, and Ant Colony System," May 2024, [Online]. Available: http://arxiv.org/abs/2405.15397

[11] B. Ghimire, A. Mahmood, and K. Elleithy, "Hybrid Parallel Ant Colony Optimization for Application to Quantum Computing to Solve Large-Scale Combinatorial Optimization Problems," Applied Sciences (Switzerland), vol. 13, no. 21, Nov. 2023, doi: 10.3390/app132111817.

[12] Z. Zheng, "Improved ant colony-based algorithm for solving traveling salesmen problem," Applied and Computational Engineering, vol. 10, no. 1, pp. 265–271, Sep. 2023, doi: 10.54254/2755-2721/10/20230191.

[13] I. Dzalbs and T. Kalganova, "Accelerating supply chains with Ant Colony Optimization across range of hardware solutions," 2020.

[14] A. A. Ismail, R. Krisnaputra, and I. Bahiuddin, "Application of Ant Colony Optimization for the Shortest Path Problem of Waste Collection Process," Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control, Aug. 2021, doi: 10.22219/kinetik.v6i3.1307.

[15] T. Dokeroglu, E. Sevinc, T. Kucukyilmaz, and A. Cosar, "A survey on new generation metaheuristic algorithms," Comput Ind Eng, vol. 137, Nov. 2019, doi: 10.1016/j.cie.2019.106040.

[16] S. Hougardy and X. Zhong, "Hard to solve instances of the Euclidean Traveling Salesman Problem," Math Program Comput, vol. 13, no. 1, pp. 51–74, Mar. 2021, doi: 10.1007/s12532-020-00184-5.

[17] B. A. de Melo Menezes, N. Herrmann, H. Kuchen, and F. Buarque de Lima Neto, "High-Level Parallel Ant Colony Optimization with Algorithmic Skeletons," Int J Parallel Program, vol. 49, no. 6, pp. 776–801, Dec. 2021, doi: 10.1007/s10766-021-00714-1.

[18] K. Tang, X. F. Wei, Y. H. Jiang, Z. W. Chen, and L. Yang, "An Adaptive Ant Colony Optimization for Solving LargeScale Traveling Salesman Problem," Mathematics, vol. 11, no. 21, Nov. 2023, doi: 10.3390/math11214439.

[19] B. A. De Melo Menezes, L. F. De Araujo Pessoa, H. Kuchen, and F. Buarque De Lima Neto, "Parallelization strategies for GPU- ased ant colony optimization applied to TSP," in Advances in Parallel Computing, IOS Press BV, 2020, pp. 321– 330. doi: 10.3233/APC200057.

[20] C. S, H. S. Seshadri, and V. Lokesha, "An Effective Parallelism Topology in Ant Colony Optimization algorithm for Medical Image Edge Detection with Critical Path Methodology (PACO-CPM)," International Journal of Recent Contributions from Engineering, Science & IT (iJES), vol. 3, no. 4, p. 12, Dec. 2015, doi: 10.3991/ijes.v3i4.5139.

[21] H. Ismkhan, "Effective heuristics for ant colony optimization to handle large-scale problems," Swarm Evol Comput, vol. 32, pp. 140–149, Feb. 2017, doi: 10.1016/j.swevo.2016.06.006.

[22] E. K. and M. S. B. A. Aslam, Multi -Threading based Implementation of Ant-Colony Optimization Algorithm for Image Edge Detection. IEEE, 2015.

[23] M. Dorigo and T. Stützle, "Ant Colony Optimization," 2004.

[24] M. Melanie, "An Introductions to Genetic Algorithms," 1999.

[25] J. Si and X. Bao, "A novel parallel ant colony optimization algorithm for mobile robot path planning," Mathematical Biosciences and Engineering, vol. 21, no. 2, pp. 2568–2586, 2024, doi: 10.3934/mbe.2024113.