 ISSN:2302-8149
e-ISSN:2540-9719

Penerapan Algoritma Cuckoo Search Berbasis Distribusi Gauss pada Permasalahan *Uncapacitated Facility Location*

Optimizing Uncapacitated Facility Location Problem with Cuckoo Search Algorithm based on Gauss Distribution

[1]Mohammad Agung Nugroho*, [2] Eto Wuryanto, [3]Kartono
[1,2,3]S1 Sistem Informasi, Fakultas Sains dan Teknologi, Universitas Airlangga
Jalan Dr. Ir. Soekarno, Mulyorejo, Surabaya, Indonesia
*e-mail: magungnugroho2@gmail.com

(***received*:** 9 Desember 2022, ***revised*:** 30 April 2023, ***accepted*:** 2 Mei 2023)

Abstrak

Penelitian ini dilakukan untuk mengetahui performa algoritma *Cuckoo Search* berbasis distribusi gauss (GCS) dalam menyelesaikan permasalahan *Facility Location Problem* (UFLP), yaitu sebuah permasalahan optimisasi dimana terdapat sejumlah *n* lokasi untuk dibangun sebuah fasilitas guna melayani sejumlah *m* pelanggan. Pada permasalahan UFLP, diasumsikan bahwa setiap fasilitas tidak memiliki batasan pelayanan, setiap pelanggan hanya dilayani oleh satu fasilitas, dan setiap lokasi hanya dapat dibangun satu fasilitas. Tujuan utama dari UFLP adalah meminimalkan biaya total pembangunan fasilitas dan biaya total pelayanan pelanggan. UFLP termasuk NP-*Hard Problem* dimana semakin besar dimensi data akan mengakibatkan perhitungan semakin kompleks. Algoritma *Cuckoo Search* merupakan algoritma yang dikembangkan dari pola tingkah laku burung *Cuckoo* dalam berkembang biak dan telah banyak dipakai untuk menyelesaikan permasalahan optimasi. Algoritma *Cuckoo Search* berbasis Distribusi Gauss (GCS) merupakan pengembangan dari algoritma *Cuckoo Search* yanng mengatasi kelemahan algoritma *Cuckoo Search* dalam waktu komputasi dan akurasi pencarian. Implementasi Algoritma GCS untuk menyelesaikan UFLP menggunakan bahasa pemprograman JavaScript dan data yang dipakai berasal dari ORLib. Hasil penelitian menunjukkan algoritma GCS mampu mencapai optimal pada semua data uji yang digunakan.

Kata kunci: Metaheuristik, Optimasi, *Cuckoo Search*, Distribusi Gauss, Penempatan Fasilitas.

Abstract

The objective of this study was to assess the capability of the Gauss distribution-based Cuckoo Search algorithm (GCS) in solving the Uncapacitated Facility Location Problem (UFLP). UFLP is an optimization problem that there are number of locations available to be built a facility so that it can serve number of customers, assuming each facility has no limits to serve customers and only a single facility is allowed to provide services to each customer. The objective function of UFLP is to minimize the combined costs of constructing facilities in an area and providing services to customers. UFLP falls under the category of NP-Hard Problems, where the computation complexity increases with the size of the data. The Cuckoo Search algorithm, which mimics the breeding behavior of Cuckoo birds, has been extensively used to tackle optimization problems. GCS was introduced to overcome the weaknesses of Cuckoo Search algorithm in terms of computational time and search accuracy. GCS used Gaussian distribution instead of Levy Flight which based on Levy distribution. In this study, the GCS algorithm was implemented using JavaScript and the dataset used was obtained from ORLib. The research outcomes showed that the GCS algorithm could achieve optimal result in all dataset.

Keywords: *Metaheuristic, Optimization, Cuckoo Search, Gauss Distribution, Facility Location.*

1 Pendahuluan

Keberhasilan sebuah bisnis bergantung pada penempatan fasilitas bisnis tersebut pada sebuah lokasi [1]. Fasilitas tersebut dapat berupa kantor, gedung, pabrik, gudang penyimpanan barang, menara pemancar, dan sebagainya. Mengoptimalkan penempatan fasilitas dapat meningkatkan

 ISSN:2302-8149
e-ISSN:2540-9719

pendapatan perusahaan, meminimalkan biaya operasional, serta dapat meningkatkan pangsa pasar dan tingkat kepuasan pelanggan. Masalah penempatan fasilitas didefinisikan sebagai masalah dimana diperlukan penempatan fasilitas yang efisien agar semua pelanggan dapat terlayani dengan biaya minimal [2]. Berbagai algoritma telah diterapkan untuk menyelesaikan *Uncapacitated Facility Location Problem* (UFLP) antara lain, *Particle Swarm Optimization* (PSO) [3], *Binary Social Spider Algorithm* (BinSSA) [4], *Scatter Search Algorithm* [5], *Crow Search Algorithm* [6], *Monkey Algorithm* [7], dan lain sebagainya.

Algoritma *Cuckoo Search* dibuat berdasarkan pola perilaku reproduksi burung Cuckoo yang meletakkan telurnya pada sarang burung lain sebagai inang. Beberapa spesies burung Cuckoo seperti *Ani* dan *Guira* akan membuang telur burung inang saat meletakkan telurnya. Burung inang terkadang dapat mengetahui keberadaan telur burung Cuckoo dalam sarang, sehingga burung inang akan membuang telur tersebut dari sarangnya atau membuat sarang baru di tempat lain yang agak jauh dari tempat sebelumnya. Beberapa spesies, misalnya Tapera, bahkan mampu meniru warna dan pola telur burung inang sehingga menurunkan kemungkinan telur burung Cuckoo dibuang [8].

Algoritma *Cuckoo Search* menurut Yang dan Deb [9] memiliki tiga aturan ideal. Pertama, burung Cuckoo meletakkan satu telur pada sarang yang dipilih secara acak sebagai inang pada satu waktu. Kedua, sarang dengan telur Cuckoo di dalamnya akan menetas dan meneruskan generasi burung Cuckoo. Ketiga, jumlah sarang tetap dan probabilitas burung inang dapat menemukan telur burung Cuckoo adalah antara 0 hingga 1. Dalam algoritma *Cuckoo Search*, solusi diasumsikan sebagai sebuah sarang.

Algoritma Cuckoo Search menggunakan *Levy Flight* untuk pencarian langkah. *Levy Flight* merupakan metode pencarian langkah yang terinspirasi dari pola pergerakan beberapa hewan dalam mencari makanan. Menurut Yang dan Deb [9], algoritma *Cuckoo Search* lebih efisien dalam mencari solusi optimal global dibandingkan dengan algoritma *Particle Swarm Optimization* dan *Genetic Algorithm*. Zeng dan Chou [10] mengusulkan algoritma Cuckoo Search dengan menggunakan *Gaussian Distribution* sebagai pengganti *Levy Flight* yang menggunakan distribusi Levy. Algoritma *Cuckoo Search* berdasarkan distribusi Gauss (GCS) diusulkan untuk mengatasi kelemahan algoritma *Cuckoo Search* yaitu *covergence speed* dan akurasi yang rendah. Dengan merujuk pada uraian sebelumnya, algoritma GCS diterapkan pada penyelesaian UFLP dengan tujuan mengevaluasi kinerja algoritma tersebut untuk mendapatkan informasi tentang performa algoritma GCS dalam menyelesaikan masalah UFLP.

2 Tinjauan Literatur

Algoritma *Cuckoo Search* terinspirasi dari perilaku burung Cuckoo dalam berkembang biak. Burung Cuckoo tidak mengerami telurnya melainkan mencari sarang burung lain lalu meletakkan telur pada sarang tersebut agar dierami oleh burung inang. Sekitar 64 dari 149 spesies burung Cuckoo [11] berkembang biak dengan cara tersebut. Burung Cuckoo cenderung memilih burung inang yang jauh berbeda dari jenisnya. Agar berhasil [8], burung Cuckoo menggunakan beberapa strategi: meniru telur burung inang; menghancurkan sarang burung lain; membuang telur burung inang; melakukan modifikasi genetik agar mempercepat waktu yang diperlukan telur untuk menetas; dan memaksa burung inang meninggalkan sarang untuk kemudian dipakai burung Cuckoo. Yang [13] kemudian menyederhanakan perilaku burung Cuckoo ke dalam tiga aturan ideal. Pertama, satu burung Cuckoo meletakkan satu telur pada sarang secara acak. Kedua, telur yang tidak dibuang burung inang (terbaik) menetas dan melanjutkan generasi baru. Ketiga, jumlah sarang burung inang tetap, dengan peluang burung inang mengenali telur burung Cuckoo p_a antara 0 hingga 1.

Algoritma *Cuckoo Search* yang diusulkan Yang dan Deb [9] menggunakan *Levy Flight*, yaitu pencarian langkah yang terinspirasi dari gerak lalat buah saat mencari makanan. Yang menyatakan bahwa *Levy Flight* lebih efektif daripada *Brownian Random Walks* dalam eksplorasi ruang pencarian yang luas.. Penelitian yang dilakukan Yang membuktikan bahwa algoritma Cuckoo Search lebih baik apabila dibandingkan dengan Algoritma Genetika dan *Particle Swarm Optimization*. Ada dua alasan utama yang menyebabkan algoritma Cuckoo Search lebih baik dari algoritma pembanding. Pertama, adanya keseimbangan antara intensifikasi dan diversifikasi. Kedua, minimnya parameter kontrol sehingga algoritma

 ISSN:2302-8149
e-ISSN:2540-9719

menjadi lebih simpel dan generik. Intensifikasi merupakan proses pencarian solusi baru di sekitar solusi terbaik sementara, sedangkan diversifikasi merupakan proses eksplorasi ruang pencarian untuk mencari global optima secara efisien, umumnya dengan menggunakan pengacakan tertentu [14].

begin

Objective function $f(x), x = (x_1, x_2, ...x_d)^T$

Generate initial population of n host nests $x_i (i = 1, 2, ..., n)$

While (t <Max Generation) or (stop criterion)

Get a cuckoo randomly by gauss distribution

Evaluate its quality/fitness F_i

Choose a nest among n(say,j)randomly

If($F_i > F_j$)

replace j by the new soluton;

End

A fraction(P_a) *of worse nests are abandoned and new ones are built;*

keep the best solutions(or nests with quality solutions);

Rank the solutions and find the current best

End while

Postprocess results and visualization

End

Gambar 1. Pseudocode Algoritma GCS

Zheng dan Zhou [10] mengemukakan algoritma *Cuckoo Search* dengan menggunakan distribusi gauss (GCS) untuk mengatasi kekurangan algoritma *Cuckoo Search* yang memiliki laju konvergensi dan akurasi yang rendah. Zheng dan Zhou mengganti *Levy Flight* dengan distribusi gauss yang disederhanakan. Eksperimen yang dilakukan Zheng dan Zhou menunjukkan performa algoritma GCS lebih baik daripada algoritma Cuckoo Search (CS). Pseudocode algoritma GCS ditampilkan pada Gambar 1. Berdasarkan eksperimen yang dilakukan Zheng dan Zhou, diharapkan algoritma GCS mampu menyelesaikan berbagai permasalahan optimasi termasuk UFLP, dengan hasil yang mendekati optimal. Penelitian ini dilakukan untuk mengetahui performa algoritma GCS pada UFLP.

3 Metode Penelitian

UFLP merupakan permasalahan dimana terdapat sejumlah m pelanggan yang harus dilayani oleh n fasilitas. Tujuan fungsi UFLP adalah mencari biaya minimum, dengan c_{ij} merupakan biaya untuk melayani pelanggan ke-i oleh fasilitas ke-j dan keputusan untuk melayani atau tidak melayani pelanggan ke-i oleh fasilitas ke-j dinotasikan dengan x_{ij}. Sementara fc_j adalah biaya pembangunan fasilitas dan y_j merupakan keputusan untuk membangun atau tidak membangun fasilitas ke-j pada sebuah lokasi. UFLP secara formal dinotasikan sebagai persamaan berikut:

$$Z = min\left(\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{j=1}^{n} fc_j y_j\right) \quad (1)$$

Dengan kendala:

$$\sum_{j=1}^{n} x_{ij} = 1 \quad i = 1, 2, \dots, m$$

 ISSN:2302-8149
e-ISSN:2540-9719

(2)

$x_{ij} \leq y_j \quad i = 1, 2, \dots, m \text{ dan } j = 1, 2, \dots, n$ (3)

$x_{ij} \geq 0 \quad i = 1, 2, \dots, m \text{ dan } j = 1, 2, \dots, n$ (4)

Data UFLP yang dipakai pada penelitian ini didapat dari ORLIB [15][16]. Data tersebut berisi biaya pembangunan fasilitas pada masing-masing lokasi, biaya pelayanan masing-masing fasilitas untuk setiap pelanggan, serta solusi optimal. Data yang didapat kemudian diolah dalam bentuk matriks m×n agar mudah dibaca dan disimpan ke dalam format *comma separated value* (.csv). Kode data yang digunakan pada penelitian ini ditampilkan dalam Tabel 1.

Tabel 1. Data yang Digunakan

Kode Data	Dimensi Data	Optimal
cap71	16×50	932615.75
cap72	16×50	977799.4
cap73	16×50	1010641.45
cap101	25×50	796648.4375
cap102	25×50	854704.2
cap103	25×50	893782.1125
cap131	50×50	793439.5625
cap132	50×50	851495.325
cap133	50×50	893076.7125

Posisi awal fasilitas didapat dengan membangkitkan bilangan acak sejumlah matriks m×n. Untuk menentukan fasilitas mana yang dibuka [3], digunakan pengkodean posisi awal menjadi angka biner 0 dan 1 sesuai persamaan (5). Apabila didapat angka 0, maka fasilitas pada lokasi tersebut tidak dibangun, sebaliknya fasilitas akan dibangun pada lokasi tersebut apabila didapat angka 1. Untuk setiap fasilitas yang dibangun, dilakukan perhitungan total biaya pembangunan fasilitas dan total biaya pelayanan sesuai fungsi tujuan pada persamaan (1). Sarang dengan total biaya terkecil selanjutnya dipilih sebagai solusi terbaik sementara.

$$y_i = \lfloor |x_i| \ (mod\ 2) \rfloor \quad (5)$$

Setelah itu, dilakukan pembaruan posisi sarang menggunakan distribusi gauss pada persamaan (6). Operator $\oplus$ merupakan *Entry-wise multiplication*. Untuk menghitung σ_s, digunakan persamaan (7) dengan σ_0 merupakan konstanta dengan nilai $\frac{1}{2}$ dan nilai $\mu = 0{,}0001$. Variabel K merupakan iterasi saat ini, sehingga hasil perhitungan σ_s akan berbeda pada setiap iterasinya. Setelah didapat posisi baru, kembali dilakukan perhitungan total biaya pembangunan dan total biaya pelayanan. Setelah itu, dilakukan perbandingan antara sarang pada posisi baru dengan sarang sebelum secara acak. Sarang dengan total biaya yang lebih kecil kemudian dipertahankan.

$$x_i^{(t+1)} = x_i^{(t)} + \alpha \oplus \sigma_s \quad (6)$$

$$\sigma_s = \sigma_0 exp(-\mu K) \quad (7)$$

Selanjutnya dicari sarang dengan kualitas buruk dengan cara membangkitkan bilangan acak antara 0 hingga 1 untuk masing-masing sarang. Apabila angka yang dibangkitkan $> P_a$, maka kualitas sarang tersebut buruk sehingga harus diganti dengan *Biased Random Walk* [17] pada persamaan (8). Sebaliknya, apabila nilai acak yang dibangkitkan $< P_a$, maka sarang dipertahankan untuk iterasi selanjutnya.

$$x_{ij}^{(t+1)} = \begin{cases} x_{ij}^{(t)} + r\,(x_{mj}^{(t)} - x_{nj}^{(t)}), & acak > P_a \\ x_{ij}^{(t)}, & acak < P_a \end{cases} \quad (8)$$

 ISSN:2302-8149
e-ISSN:2540-9719

Biased Random Walk digunakan untuk mencari posisi baru di sekitar posisi sarang saat ini. Variabel $x_{mj}^{(t)}$ dan $x_{nj}^{(t)}$ merupakan posisi sarang m dan sarang n yang dipilih secara acak dari sekumpulan populasi sarang X_n pada iterasi t. Variabel r merupakan angka acak antara 0 hingga 1. Setelah didapat posisi baru untuk setiap sarang dengan kualitas buruk, dilakukan perhitungan total biaya dan dilakukan perbandingan total biaya. Apabila total biaya sarang baru lebih besar dari sarang sebelum, maka posisi sarang sebelum dipertahankan untuk iterasi berikutnya. Sebaliknya, apabila total biaya sarang baru lebih kecil daripada sarang sebelum, maka posisi sarang sebelum digantikan dengan posisi sarang baru hasil *Biased Random Walk*.

$$t_{length} = \lceil t_{maks} \times limit \rceil \tag{9}$$

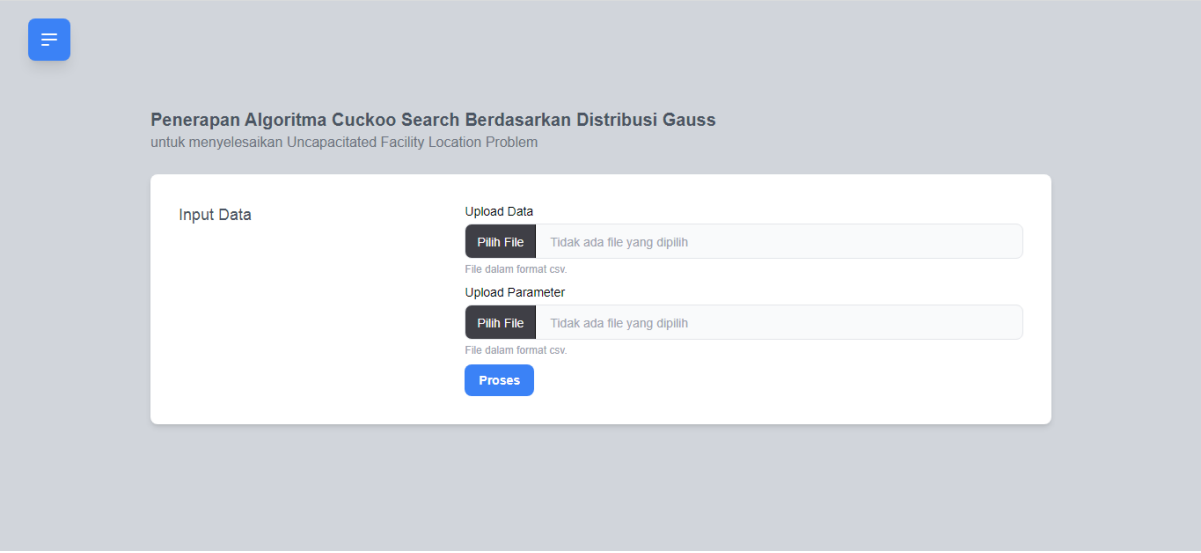
Dengan,

$$limit = \begin{cases} 0.05, & t_{maks} \geq 1000 \\ 1, & t_{maks} < 1000 \end{cases} \tag{10}$$

Penelitian ini menggunakan 2 kriteria stop: (1) apabila iterasi telah mencapai t_{maks} yang sudah ditentukan sejak awal; (2) apabila total biaya yang didapat adalah sama selama t_{length} iterasi yang ditentukan secara adaptif menggunakan persamaan (9). Nilai variabel $limit$ dapat diubah-ubah sesuai kebutuhan. Selanjutnya dilakukan implementasi algoritma pada program komputer berbasis web dengan menggunakan bahasa pemprograman JavaScript.

4 Hasil dan Pembahasan

Dalam eksperimen ini, digunakan komputer dengan spesifikasi prosesor AMD A8-4500M dan RAM sebesar 8 GB, yang berjalan dalam sistem operasi Windows 8.1. Percobaan ini menggunakan program komputer berbasis web yang disusun dengan bahasa pemprograman JavaScript dan dijalankan dengan bantuan *browser* Google Chrome. Gambar 2 menunjukkan tampilan program komputer yang digunakan pada percobaan ini.



Gambar 2. Tampilan Program Komputer

Program komputer tersebut disusun berdasarkan *pseudocode* yang disajikan pada Gambar 1. Pengguna akan diminta untuk memasukkan data dan parameter percobaan, dalam format *file comma separated value* (csv). Perhitungan akan dimulai setelah pengguna menekan tombol proses. Setelah proses perhitungan selesai, pengguna dapat melihat hasil perhitungan dengan menekan tombol hasil

 ISSN:2302-8149
e-ISSN:2540-9719

seperti yang ditunjukkan pada Gambar 3. Hasil perhitungan berupa total biaya minimum yang didapat pada setiap iterasi sebanyak jumlah perulangan percobaan dalam bentuk *file* html.

Gambar 3. Tampilan Hasil Program Komputer

Pada masing-masing data dilakukan percobaan sebanyak 30 kali dengan iterasi maksimum ditentukan sebanyak 25.000 iterasi, jumlah populasi sarang ditentukan sebanyak 250 sarang [9], probabilitas telur burung Cuckoo ditemukan sebesar 0,25 [18], dan alpha yang digunakan sebesar 0,01. Rekapitulasi hasil percobaan pada masing-masing data disajikan pada tabel 2 berikut.

Tabel 2. Rekapitulasi Hasil Percobaan Setiap Data

Data	ARPE	*Hit*	*Fit Best*	*Fit Worst*	*Fit Avg*	*Std. Dev*	ACPT
cap71	0.000	30	932615.75	932615.75	932615.75	0.000	369
cap72	0.000	30	977799.40	977799.40	977799.40	0.000	334
cap73	0.000	30	1010641.45	1010641.45	1010641.45	0.000	251
cap101	0.237	5	796648.44	801858.5	798533.32	1525.21	734
cap102	0.214	9	854704.2	861114.96	856529.13	1910.23	683
cap103	0.185	9	893782.11	900621.9	895437.32	2144.28	631
cap131	1.253	1	793439.56	829196.1	803384.49	9916.76	1342
cap132	1.755	1	851495.33	909141.28	866438.49	18403.19	1018
cap133	2.256	1	893076.71	980008	913223.27	27446.10	1036

ARPE [3] merupakan *average relative percent error* yang dihitung menggunakan persamaan (11). ARPE merupakan rata-rata kedekatan hasil yang didapat dengan nilai optimal dikalikan 100. Sementara *hit* merupakan banyaknya hasil yang merupakan nilai optimal yang didapat dalam 30 kali percobaan. *Fit best* merupakan hasil terbaik yang mampu dicapai algoritma dalam 30 percobaan, sementara *fit worst* merupakan hasil terburuk yang dihasilkan oleh perhitungan algoritma dalam 30 kali percobaan. *Fit avg* merupakan rata-rata nilai yang didapat, sementara *Std. Dev* merupakan simpangan baku. *Average computational processing time* (ACPT) merupakan rata-rata lamanya waktu pemprosesan data menggunakan algoritma dalam satuan detik.

$$ARPE = \sum_{i=1}^{R} \left(\frac{f_i - f_{opt}}{f_{opt}} \right) \times \frac{100}{R} \quad (11)$$

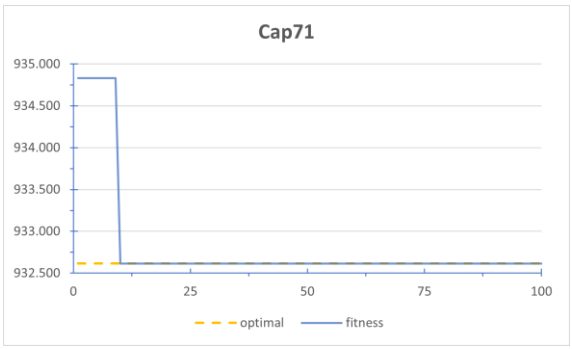
R merupakan banyaknya percobaan yang dilakukan. Sementara f_i merupakan hasil yang didapat pada percobaan tersebut, sedangankan f_{opt} merupakan hasil optimal dari data. Semakin kecil nilai ARPE, maka solusi yang dihasilkan semakin mendekati optimal. Sementara itu, semakin besar nilai hit menandakan seberapa bagus dan konsisten sebuah algoritma dalam mencapai hasil optimal.

 ISSN:2302-8149
e-ISSN:2540-9719

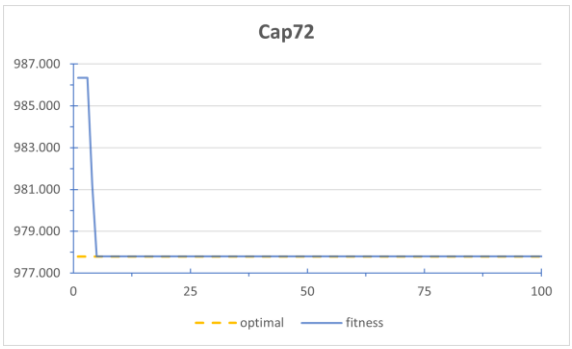
Dari 9 buah data yang diujikan, algoritma GCS terbukti mampu mencapai optimal setidaknya 1 kali. Untuk data berukuran 16×50, algoritma GCS secara konsisten mampu mencapai optimal dengan nilai ARPE sebesar 0 dan *hit* 30 kali. Sementara pada data berukuran 25×50, performa algoritma GCS mulai terlihat kurang stabil. Hal ini dibuktikan dengan ARPE yang didapat berkisar 0.1 sampai 0.2 dengan jumlah *hit* paling banyak adalah 9 kali. Pada data berukuran 50×50, performa algoritma GCS cukup buruk dengan ARPE yang didapat berkisar 1 sampai 3 dan hanya mampu mencapai nilai optimal sebanyak 1 kali dari 30 kali percobaan. Berdasarkan hasil percobaan, performa algoritma GCS akan menurun seiring bertambahnya dimensi data. Sehingga, algoritma GCS kurang cocok digunakan untuk menghitung UFLP dengan dimensi data yang besar.

Performa terbaik algoritma GCS pada data kecil dengan kode Cap71 yang disajikan pada gambar 4 menghasilkan nilai optimal pada iterasi ke-10 dengan nilai *Fit* sebesar 932.615,75. Hasil tersebut didapat pada percobaan ke-21. Sementara pada data Cap72, algoritma GCS menghasilkan nilai optimal pada iterasi ke-7 pada percobaan ke-20 dengan nilai *Fit* sebesar 977.799,40. Performa terbaik algoritma GCS pada data Cap72 ditunjukkan pada gambar 5. Pada gambar 6 disajikan performa terbaik algoritma GCS pada data Cap73 yang dihasilkan pada percobaan ke-4 di iterasi ke-18 dengan nilai *Fit* sebesar 1.010.641,45.

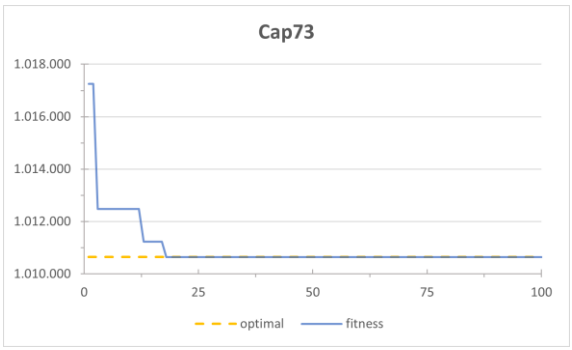
Untuk data sedang dengan kode Cap102 pada gambar 7, performa terbaik didapat pada percobaan ke-14 dengan nilai *Fit* 796.648,44. Nilai tersebut didapat pertama kali pada iterasi ke-1.685 dan program berhenti pada iterasi 2.934 dengan waktu proses 772 detik. Sementara itu, pada data dengan kode Cap102 yang ditunjukkan pada gambar 8 didapat nilai *Fit* sebesar 854.704,2 pada percobaan ke-6. Nilai tersebut didapatkan pertama kali pada iterasi ke-1.636 hingga program berhenti pada iterasi 2.885 dengan waktu proses 610 detik. Pada data Cap103 yang disajikan pada gambar 9, didapat nilai *Fit* sebesar 893.782,11 pada percobaan ke-20. Nilai tersebut didapatkan pertama kali pada iterasi ke-847 dan program berhenti pada iterasi 2.096 dengan waktu proses sebesar 532 detik.



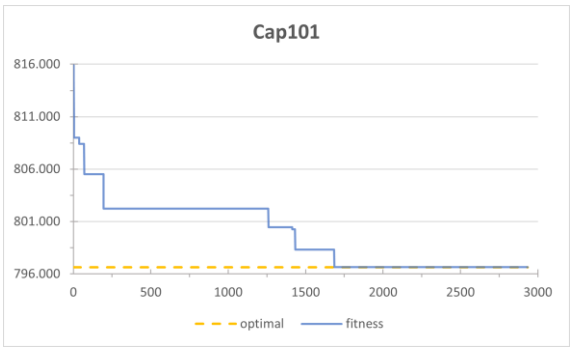
Gambar 4. Performa GCS pada cap71



Gambar 5. Performa GCS pada cap72

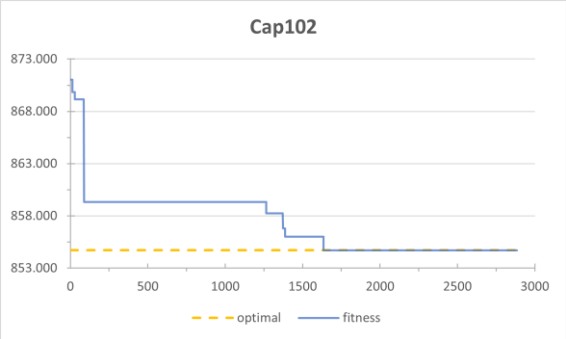


Gambar 6. Performa GCS pada cap73

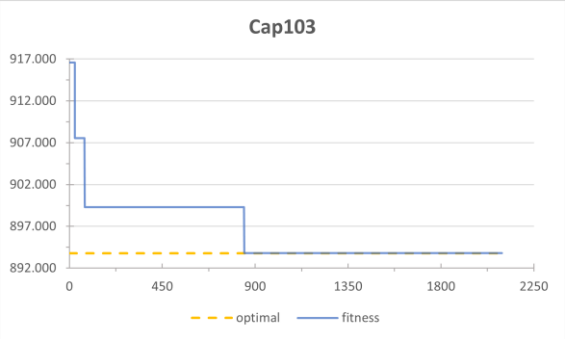


Gambar 7. Performa GCS pada cap102

 ISSN:2302-8149
e-ISSN:2540-9719

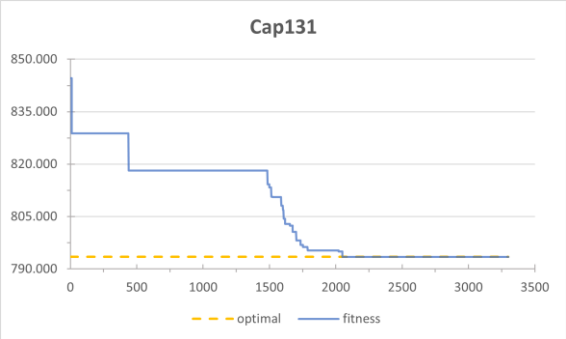


Gambar 8. Performa GCS pada cap101

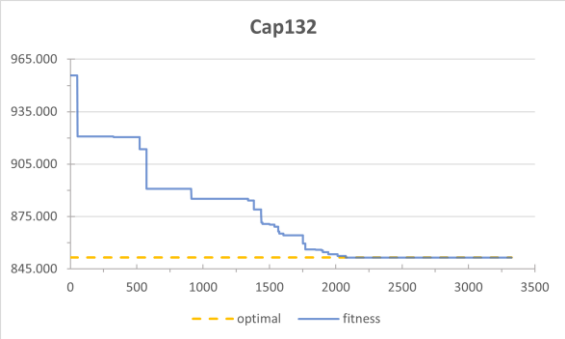


Gambar 9. Performa GCS pada cap102

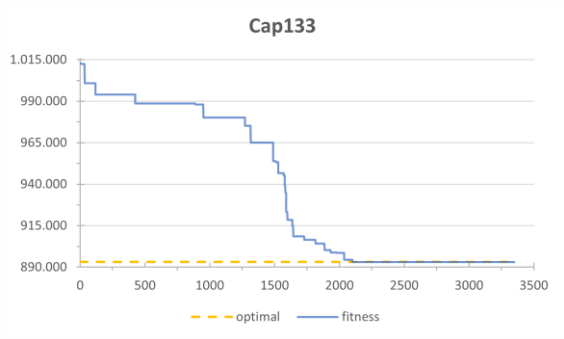
Sementara pada data besar dengan kode Cap131 yang ditunjukkan pada gambar 10, performa terbaik didapat pada percobaan ke-5 dengan nilai *Fit* 793.439,56. Nilai tersebut didapat pertama kali pada iterasi ke-2.051 dan program berhenti pada iterasi 3.300 dengan waktu proses 1.186 detik. Pada data dengan kode Cap132 yang ditunjukkan pada gambar 11 didapat nilai *Fit* sebesar 851.495,33 pada percobaan ke-23. Nilai tersebut didapatkan pertama kali pada iterasi ke-2.076 hingga program berhenti pada iterasi 3.325 dengan waktu proses 1.066 detik. Pada data Cap133 yang disajikan pada gambar 12, didapat nilai *Fit* sebesar 893.076,71 pada percobaan ke-9. Nilai tersebut didapatkan pertama kali pada iterasi ke-2.101 dan program berhenti pada iterasi 3.350 dengan waktu proses sebesar 1.083 detik.



Gambar 10. Performa GCS pada cap131



Gambar 11. Performa GCS pada cap132



Gambar 12. Performa GCS pada cap133

5 Kesimpulan

Penelitian ini dilakukan untuk mengetahui performa algoritma GCS untuk menyelesaikan permasalahan UFL. Berbeda dari algoritma Cuckoo Search yang menggunakan Levy Flight dalam pencarian solusi, algoritma GCS menggunakan distribusi gauss yang telah disederhanakan. Algoritma GCS diusulkan untuk mengatasi kelemahan algoritma Cuckoo Search dalam hal akurasi dan kecepatan konvergensi yang rendah.

Berdasarkan percobaan yang telah dilakukan, dapat disimpulkan bahwa algoritma GCS mampu mencapai nilai optimal pada UFLP. Performa algoritma GCS pada data UFLP dengan dimensi 16×50

sangat baik dan konsisten menghasilkan hasil optimal. Sementara pada data UFLP dengan dimensi 25×50, performa algoritma GCS dapat dikatakan baik karena mampu mencapai hasil optimal 9 kali dari 30 kali percobaan. Untuk data UFLP dengan dimensi 50×50, algoritma GCS cukup buruk. Hal ini dibuktikan dengan tingginya ARPE dan simpangan baku solusi yang dihasilkan. Performa algoritma GCS pada data UFLP dengan ukuran besar masih dapat ditingkatkan, misalnya dengan melakukan modifikasi pada algoritma atau dengan menggunakan operator tertentu, sehingga membutuhkan penelitian lebih lanjut.

References

[1] S. Madavi and P. B. Sangode, "Exploring the Influence of Facility Location on the Operations of Service Organizations," *IOSR J. Eng.*, vol. 9, no. 5, pp. 45–52, 2019.
[2] L. A. W. Cornuejols, Gerard ; George L. Nemhauser, "The Uncapicitated Facility Location Problem." 1984.
[3] A. R. Guner and M. Sevkli, "A Discrete Particle Swarm Optimization Algorithm for Uncapacitated Facility Location Problem," *J. Artif. Evol. Appl.*, vol. 2008, pp. 1–9, Apr. 2008.
[4] E. Baş and E. Ülker, "A binary social spider algorithm for uncapacitated facility location problem," *Expert Syst. Appl.*, vol. 161, 2020.
[5] H. Hakli and Z. Ortacay, "An improved scatter search algorithm for the uncapacitated facility location problem," *Comput. Ind. Eng.*, vol. 135, no. June, pp. 855–867, 2019.
[6] E. Sonuç, "Binary crow search algorithm for the uncapacitated facility location problem," *Neural Comput. Appl.*, vol. 33, no. 21, pp. 14669–14685, 2021.
[7] S. Atta, P. R. S. Mahapatra, and A. Mukhopadhyay, *Solving uncapacitated facility location problem using monkey algorithm*, vol. 695. Springer Singapore, 2018.
[8] E. M. Fitzpatrick-Wacker, "Cuckoo Brood Parasitism," in *Encyclopedia of Animal Cognition and Behavior*, no. Langmore 2013, Cham: Springer International Publishing, 2020, pp. 1–5.
[9] X.-S. Yang and Suash Deb, "Cuckoo Search via Lévy flights," in *2009 World Congress on Nature & Biologically Inspired Computing (NaBIC)*, 2009, pp. 210–214.
[10] H. Zheng and Y. Zhou, "A novel Cuckoo Search optimization algorithm base on gauss distribution," *J. Comput. Inf. Syst.*, vol. 8, no. 10, pp. 4193–4200, 2012.
[11] O. Kr, *Avian Brood Parasitism*, no. July. Cham: Springer International Publishing, 2017.
[12] X.-S. S. Yang and M. Karamanoglu, *Nature-Inspired Metaheuristic Algorithms Second Edition*, Second Edi. Frome: Luniver Press, 2010.
[13] I. Fister, X.-S. Yang, D. Fister, and I. Fister, "Cuckoo Search: A Brief Literature Review," in *Studies in Computational Intelligence*, vol. 516, 2014, pp. 49–62.
[14] X. S. Yang and S. Deb, "Engineering optimisation by cuckoo search," *Int. J. Math. Model. Numer. Optim.*, vol. 1, no. 4, pp. 330–343, 2010.
[15] J. E. Beasley, "Or-library: Distributing test problems by electronic mail," *J. Oper. Res. Soc.*, vol. 41, no. 11, pp. 1069–1072, 1990.
[16] J. E. Beasley, "No Title," 1990. [Online]. Available: http://people.brunel.ac.uk/~mastjjb/jeb/info.html.
[17] Y. Lin, C. Zhang, and Z. Liang, "Cuckoo Search Algorithm with Hybrid Factor Using Dimensional Distance," *Math. Probl. Eng.*, vol. 2016, pp. 1–11, 2016.
[18] Q. Pan, C. Darabos, and J. H. Moore, *Cuckoo Search and Firefly Algorithm*, vol. 516. Cham: Springer International Publishing, 2014.