OPTIMIZING NEURAL NETWORK CLASSIFIER FOR DIABETES DATA USING METAHEURISTIC ALGORITHMS

Ajie Pangestu[1], Berton Arie[2], Elia Jovi[3], Rina Dwi S[4], Heri Prasetyo[5]
Program Studi Informatika, Fakultas Matematika dan Ilmu Pengetahuan Alam
Universitas Sebelas Maret Surakarta, Jawa Tengah, Indonesia
E-mail: [1]pangestu@student.uns.ac.id, [2]bertonarie123@student.uns.ac.id, [3]elia.jm@student.uns.ac.id,
[4]rina_setianingrum@student.uns.ac.id, [5]heri.prasetyo@staff.uns.ac.id

Abstract – This paper reports the performance comparison among several metaheuristics algorithms on the neural network *training. In this research we use five metaheuristic algorithms which implements for diabetes data, there are Particle Swarm Optimizer (PSO), Multi-Verse Optimizer (MVO), Grey Wolf Optimizer (GWO), Bat Algorithm (BAT), and Cuckoo Search (CS). The Cuckoo Search (CS) algorithm is a recently developed meta-heuristic optimization algorithm which is suitable for solving optimization problems. The main problem to be solved is to find the most effective meta-heuristic optimization algorithm. The search was done by comparing the results of PSO (Particle Swarm Optimizer) algorithm test with the test with MVO (Multi-Verse Optimizer), GWO (Grey Wolf Optimizer), BAT (Bat Algorithm), and CS (Cuckoo Search). Then look for the most effective algorithm. The best metaheuristic algorithm that we had in this research is MVO, with best case accuracy result 78% and lowest standard deviation is 0.00675 and the worst is BAT algorithm with best case accuracy 77% and standard deviation 0.14571.*

Keywords: Jaringan Saraf Tiruan, PSO, MVO, GWO, BAT, CS, *meta-heuristic optimization algorithm*

I. PENDAHULUAN

LATAR BELAKANG

Teknik optimasi meta-heuristik telah menjadi sangat populer selama dua periode terakhir [1]. Terdapat beberapa algoritma, diantaranya seperti PSO [2], MVO [3], GWO [1], BAT [4], dan CS [5].

Selain banyaknya karya teoritis, teknik pengoptimalan seperti ini telah diterapkan di berbagai bidang pembelajaran. Terdapat pertanyaan disini tentang mengapa meta-heuristik menjadi sangat umum. Jawaban pertanyaan tersebut diringkas menjadi empat alasan utama: kesederhanaan, fleksibilitas, mekanisme bebas derivasi, dan penghindaran optima local [1].

Pertama, meta-heuristik cukup sederhana. Sebagian besar terinspirasi oleh konsep yang sangat sederhana. Inspirasi biasanya terkait dengan fenomena fisik, perilaku hewan, atau konsep evolusioner. Kesederhanaan memungkinkan ilmuwan komputer untuk mensimulasikan konsep alam yang berbeda, mengusulkan meta-heuristik baru, menghibridisasi dua atau lebih meta-heuristik, atau memperbaiki meta-heuristik saat ini. Apalagi kesederhanaannya membantu ilmuwan lain untuk belajar meta-heuristik dengan cepat dan menerapkannya pada masalah mereka.

Kedua, fleksibilitas mengacu pada penerapan meta-heuristik terhadap berbagai masalah tanpa adanya perubahan khusus dalam struktur algoritma. Meta-heuristik mudah diterapkan pada berbagai masalah karena kebanyakan menganggap masalah sebagai kotak hitam. Dengan kata lain, hanya *input* dan *output* dari sebuah sistem yang penting untuk meta-heuristik. Jadi, semua kebutuhan perancang adalah untuk mengetahui bagaimana mempresentasikan masalahnya bagi meta-heuristik.

Ketiga, mayoritas meta-heuristik memiliki mekanisme bebas derivasi. Berbeda dengan pendekatan optimasi berbasis gradien, masalah optimisasi meta-heuristik secara stokastik. Proses optimasi dimulai dengan solusi acak, dan tidak perlu menghitung turunan dari ruang pencarian untuk mencari yang optimal. Hal ini membuat meta-heuristik sangat sesuai untuk masalah nyata dengan informasi derivatif yang mahal atau tidak diketahui.

Akhirnya, meta-heuristik memiliki kemampuan superior untuk menghindari optima lokal dibandingkan dengan teknik pengoptimalan konvensional. Hal ini disebabkan oleh karakteristik stokastik meta-heuristik yang memungkinkan mereka menghindari stagnasi solusi lokal dan mencari keseluruhan ruang pencarian secara ekstensif. Ruang pencarian masalah sebenarnya biasanya tidak diketahui dan sangat kompleks dengan sejumlah besar optima lokal, jadi meta-heuristik adalah pilihan bagus untuk mengoptimalkan masalah nyata yang menantang ini.

Secara umum, meta-heuristik dapat dibagi menjadi dua kelas utama: berbasis solusi tunggal dan basis populasi. Solusi kandidat tunggal ini kemudian ditingkatkan selama iterasi. Populasi berbasis meta-heuristik, bagaimanapun, melakukan optimasi dengan menggunakan seperangkat solusi (populasi). Dalam kasus ini, proses pencarian dimulai dengan populasi awal acak (multiple solutions), dan populasi ini ditingkatkan selama iterasi. Meta-heuristik berbasis populasi memiliki beberapa keunggulan dibandingkan dengan algoritma berbasis solusi tunggal [1]:

- Beberapa solusi kandidat berbagi informasi tentang ruang pencarian yang menghasilkan lompatan mendadak ke bagian pencarian ruang pencarian yang menjanjikan.
- Beberapa solusi kandidat saling membantu untuk menghindari solusi yang optimal secara lokal.
- Populasi berbasis meta-heuristik umumnya memiliki eksplorasi lebih besar dibandingkan dengan algoritma solusi berbasis tunggal.

Dari penjelasan di atas, dalam penelitian ini akan digunakan beberapa algoritma optimasi meta-heuristik yang kemudian hasil pengujian dengan masing-masing algoritma akan dibandingkan dan diperoleh algoritma yang paling efektif.

II. DASAR TEORI

Di dalam bab ini akan dijelaskan pengertian mengenai masing-masing algoritma.

1. PSO (*Particle Swarm Optimizer*)

PSO mensimulasikan perilaku burung berkelompok. Misalkan skenario berikut: Sekelompok burung yang secara acak mencari makanan di suatu daerah. Hanya ada satu potong makanan di daerah yang dicari. Semua burung tidak tahu di mana makanan ini. Tapi mereka tahu berapa banyak makanan di setiap iterasi. Jadi apa strategi terbaik untuk mencari makanan? Cara efektif adalah dengan mengikuti burung yang paling dekat dengan makanan. PSO belajar dari skenario dan menggunakannya untuk memecahkan masalah optimasi. Dalam PSO, setiap solusi tunggal merupakan "sebuah burung" dalam ruang pencarian. Kami menyebutnya "partikel". Semua partikel memiliki nilai *fitness* yang dievaluasi oleh fungsi *fitness* harus dioptimalkan dan memiliki kecepatan penerbangan langsung dari partikel. Partikel-partikel terbang melalui ruang masalah dengan mengikuti partikel optimal saat ini [6].

Algoritma PSO menggunakan dua variabel random r_1 dan r_2 yang keduanya menghasilkan angka acak dengan jarak antara 0 dan 1. Variabel random tersebut digunakan untuk memberikan efek kepada sifat *stochastic* dari algoritma PSO. Nilai dari r_1 dan r_2 disesuaikan dengan konstanta c_1 dan c_2 yang memiliki rentang nilai antara $0 < c_1, c_2 \leq 2$. Konstanta tersebut disebut sebagai koefisien akselerasi yang mempengaruhi jarak maksimum yang dapat diambil oleh sebuah partikel dalam sebuah iterasi. *Update* kecepatan dari sebuah partikel dibedakan untuk setiap dimensi $j \in 1 \ldots n$ (n berdasarkan jumlah parameter yang dioptimasi), sehingga $v_{i,j}$ mewakili dimensi ke j dari vektor kecepatan yang diasosiasikan dengan partikel ke i. Sehingga persamaan untuk update kecepatan dapat didefinisikan sebagai berikut:

$$vi = \omega\, vi + c_1 r_1\, (pbest - xi) + c_2 r_2\, (gbest - xi)$$

Dimana:

vi = kecepatan partikel saat ini

ώ = berat *inertia*

c_1, c_2 = koefisien akselerasi

r_1, r_2 = bilangan random uniform antara 0 dan 1

$pbest$ = posisi personal best partikel saat ini

$gbest$ = posisi global best partikel saat ini

xi = posisi partikel saat ini

Dapat dilihat dari persamaan *update* kecepatan bahwa c_2 mengatur jarak maksimum yang dipengaruhi oleh partikel global best ($gbest$), dan c_1 mengatur jarak yang dipengaruhi oleh posisi personal best dari partikel tersebut. Nilai dari vi dapat dibatasi dengan nilai $[-vmax, vmax]$ untuk mencegah terjadinya kejadian dimana partikel meninggalkan daerah pencarian. Jika daerah pencarian dibatasi dengan $[-xmax, xmax]$, maka nilai $vmax$ biasanya didefinisikan sebagai:

$$vmax = k\, xmax, dimana\ 0.1 \leq k \leq 1.0$$

Posisi setiap partikel di-*update,* sehingga dihasilkan persamaan berikut:

$$xi = xi + vi$$

dimana:

xi = posisi partikel baru

x = posisi partikel saat ini

vi = kecepatan partikel yang baru

2. MVO (*Multi-Verse Optimizer*)

Algoritma MVO (*Multi-Verse Optimizer*) adalah salah satu algoritma optimasi yang dapat digunakan untuk pengambilan keputusan. Contoh yang dibahas kali ini adalah mengenai pencarian posisi dengan pengembalian nilai fungsi maksimal.

Algoritma ini menggunakan topik perpindahan posisi alam semesta atau dinamakan dengan *universe*. Terdapat 2 macam cara perpindahan posisi *universe*, yaitu melalui proses perpindahan materi dari *Whitehole* ke *Blackhole*, dan melalui proses perpindahan berkecepatan cahaya yang dilakukan dalam ruang *Wormhole* [3].

3. GWO (*Grey Wolf Optimizer*)

Algoritma GWO (*Grey Wolf Optimizer*) adalah salah satu algoritma optimasi yang dapat digunakan untuk pengambilan keputusan. Contoh yang dibahas kali ini adalah mengenai pencarian posisi dengan pengembalian nilai fungsi maksimal [1].

Algoritma ini meniru tingkah laku dari spesies *Grey Wolf* dalam berburu. Proses utama *Grey Wolf* dalam berburu secara berkelompok dikonversi ke dalam bentuk matematika dan digunakan untuk memecahkan permasalahan optimasi.

4. BAT (*Bat Algorithm*)

Algoritma BAT (*Bat Algorithm*) adalah salah satu algoritma optimasi yang dapat digunakan untuk pengambilan keputusan. Contoh yang dibahas kali ini adalah mengenai pencarian posisi dengan pengembalian nilai fungsi maksimal. Algoritma ini meniru tingkah laku dari kelelawar yang berpindah-pindah tempat menggunakan pancaran getaran untuk mengetahui keadaan sekitar. Kelelawar akan cenderung memilih tempat yang lebih sunyi dibandingkan tempat yang bising [4].

5. CS (*Cuckoo Search*)

Cuckoo Search merupakan algoritma heuristik modern yang didasarkan pada perilaku burung *cuckoo*. Burung *cuckoo* memiliki strategi reproduksi yang unik, dimana mereka akan menaruh telur mereka di sarang burung jenis lain [5]. Setiap burung *cuckoo* hanya menaruh satu butir telurnya di sarang burung lain yang dipilih secara acak. Sarang burung yang menghasilkan

generasi *cuckoo* terbaik akan melanjutkan proses ke generasi berikutnya. Setiap pergantian generasi, jumlah dari pemilik sarang burung asli akan diatur kembali dan pemilik asli sarang ini mempunyai peluang untuk mengenali telur burung *cuckoo* yang ditaruh di sarangnya dengan probabilitas 0 sampai 1 (Pa). Dalam hal ini, bila pemilik asli sarang menemukan telur burung *cuckoo*, maka pemilik itu dapat memilih untuk membuang telur tersebut ataupun meninggalkan sarangnya. Ketika sedang menghasilkan generasi baru $x\ (t\ +\ 1)$, maka akan digunakan proses pengacakan langkah dengan *Lévy Flights* yang dapat dilihat sebagai berikut:

$$xi^{(t+1)} = xi^{(t)} + a \oplus Levy\ (\lambda) \quad \text{..... (1)}$$

di mana $\alpha > 0$ merupakan ukuran langkah yang dikaitkan dengan tingkatan masalah yang dikerjakan. Sedangkan $Lévy\ (\lambda)$ menyatakan fungsi persamaan posisi dari *Lévy Flights*, yang bentuk persamaannya adalah sebagai berikut:

$$Levy \sim u = t^{-\lambda}, (1 < \lambda \leq 3) \quad \text{..... (2)}$$

III. METODOLOGI

Di dalam metologi, dijelaskan mengenai algoritma dari masing-masing. Berikut adalah penjelasannya:

1. Algoritma PSO

Ada beberapa prosedur yang harus dilakukan untuk menerapkan Algoritma PSO dalam menyelesaikan masalah [7]:

a. Inisiasi partikel dan membangkitkan kecepatan secara random
Inisialisasi partikel dapat menggunakan pengkodean yang bertujuan menyederhanakan masalah. Dalam pengkodean masalah, [8] menggunakan angka permutasi dalam merepresentasikan nilai dimensi dan selanjutnya dimensi tersebut akan dikonversi kedalam indeks bahan makanan. Konversi nilai dimensi menjadi Indeks Bahan Makanan ditunjukkan pada Persamaan (1).

$$Indeks\ dimensi\ ke - 1 = ((a-1)\ (\tfrac{b-1}{c-1})) + 1 \quad \text{..... (1)}$$

Keterangan:
a = jumlah anggota jenis bahan makanan ke-i
$\boldsymbol{b}$ = nilai dimensi ke-i
c = batas atas angka permutasi

b. Evaluasi partikel dengan cara membandingkan nilai *fitness* Nilai *fitness* digunakan sebagai acuan untuk menentukan Gbest dan Pbest.

c. Mencari Pbest
Mencari Pbest dilakukan dengan cara membandingkan nilai Pbest sebelum dan sesudah Iterasi. Jika nilai *fitness* partikel baru lebih besar dari *fitness* Pbest sebelumnya maka partikel tersebut dijadikan Pbest terbaru.

d. Mencari Gbest sebagai partikel terbaik dari seluruh anggota *swarm*
Nilai Gbest didapatkan dari nilai fitness Pbest tertinggi.

e. Memperbarui Kecepatan (*Velocity*) dan posisi partikel Nilai *velocity* didapatkan dari penjumlahan momentum dan pengalaman yang diambil dari Gbest dan Pbest. Momentum didapatkan dengan cara mengkalikan bobot inersia dan kecepatan sebelumnya. Untuk menentukan nilai *velocity* dapat melihat Persamaan (2). Untuk menghitung bobot inersia dapat dilihat pada Persamaan (3).

Persamaan 2:

$${}^{k}_{j}V = w\,{}^{k}_{j}V + c_1 rand_1\ x\ (Pbest_j - {}^{k}_{j}x) + c_2 rand_2\ x\ (Gbest_j - {}^{k}_{j}x)$$

Persamaan 3:

$$W = Wmax - \frac{Wmax - Wmin}{iter\ max}\ x\ iter$$

Dimana:
${}^{k}_{j}V$ = *velocity* dimensi ke-j pada *iterasi* ke-k
W = bobot *inertia*
c_1 = nilai koefisien akselerasi ke-1
c_2 = nilai koefisien akselerasi ke-2
$rand$ = nilai random $[0, 1]$
${}^{k}_{j}X$ = posisi dimensi ke-j pada *iterasi* ke-k
$Pbest_j$ = nilai $Pbest$ dari dimensi ke-j

Posisi terbaru dapat diperoleh dari hasil penjumlahan Posisi sebelumnya dengan Kecepatan baru. Menghitung posisi terbaru menggunakan Persamaan (4) dibawah ini:

$${}^{k+1}_{j}X = {}^{k}_{j}X + {}^{k+1}_{j}V \quad \text{..... (4)}$$

Melanjutkan langkah ke-2 jika *stopping condition* belum terpenuhi

Untuk mengetahui algoritma PSO dapat dilihat dari *pseudocode* yang dijelaskan pada gambar 3.1.

```
for each particle i = 1, ..., S do
    Initialize the particle's position with a uniformly distributed
    random vector: xi ~ U(bLo, bup)
    Initialize the particle's best known position to its initial
    position: pi + xi
    if f(pi) < f(g) then
        update the swarm's best known position: g + pi
    Initialize the particle's velocity: vi ~U(-|bup-bLo|, |bup-bLo|)
while a termination criterion is not met do:
    for each particle i = 1, ..., S do
        for each dimension d = 1, ..., n do
        Pick random numbers: rp, rg ~U(0,1)
            Update the particle's velocity: vi,d+? vi,d+fp rp
    (pi,d-xi,d)+fg rg (gd-xi,d)
        Update the particle's position: xi-xi+vi
        if f(xi) < f(pi) then
            Update the particle's best known position: pi-xi
            if f(pi) < f(g) then
                Update the swarm's best known position: g-pi
```

Gambar 3. 1 *Pseudocode* Algoritma PSO [2]

2. Algoritma MVO

2.1. Inspirasi

Teori Big Bang membahas bahwa alam semesta kita dimulai dengan ledakan besar. Menurut teori ini, besar Bang adalah asal

segala sesuatu di dunia ini, dan memang ada tidak ada sebelum itu Beberapa alam semesta berinteraksi dan bahkan mungkin bertabrakan satu sama lain dalam *multi-verse optimization*.

2.2. Pencarian

Untuk mengetahui algoritma MVO dapat dilihat dari *pseudocode* yang dijelaskan pada gambar 3.2.

```
for each universe indexed by i
    for each object indexed by j
        r2=random([0,1]);
        if r2<Wormhole_existance_probability
            r3=random([0,1]);
            r4=random([0,1]);
            if r3<0.5

        U(i,j)=Best_universe(j)+Travelling_distance
_rate*((ub(j)-lb(j))*
                r4+lb(j));
            else
                U(i,j)=Best_universe(j)-
Travelling_distance_rate*((ub(j)-lb(j))*
                r4+lb(j));
            end if
        end if
        end for
    end for
```

Gambar 3. 2 *Pseudocode* Algoritma MVO [3]

3. Algoritma GWO

3.1. Inspirasi

Serigala abu-abu (Canis lupus) milik keluarga *Canidae*. Serigala abu-abu dianggap sebagai predator puncak, yang berarti bahwa mereka berada di puncak dari rantai makanan. Serigala abu-abu kebanyakan lebih suka tinggal di dalam satu pak. Itu Ukuran kelompok rata-rata 5-12. Yang menarik adalah mereka memiliki hirarki yang dominan sosial yang sangat ketat seperti yang ditunjukkan pada gambar 3.1. Pemimpinnya adalah laki-laki dan perempuan, yang disebut alfa. Alpha adalah sebagian besar bertanggung jawab untuk membuat keputusan tentang berburu, tempat untuk tidur, waktu bangun, dan sebagainya.

Gambar 3. 3 Hirarki Algoritma GWO

Tingkat kedua dalam hirarki *grey wolf* adalah beta. *Beta wolf* bisa berupa jantan atau betina, dan dia mungkin kandidat terbaik untuk menjadi alfa dalam kasus salah satu *alpha wolve* meninggal dunia atau menjadi sangat tua. *Beta wolf* harus menghormati alfa, tapi memerintahkan tingkat rendah lainnya serigala

Dalam peringkat terendah *grey wolf* adalah omega. Omega memainkan peran kambing hitam. *Omega wolf* selalu harus tunduk pada semua serigala dominan lainnya.

Jika serigala bukan alfa, beta, atau omega, dia disebut subordinate (atau delta dalam beberapa referensi). *Delta wolf* harus tunduk pada alpha dan beta, tapi mereka mendominasi omega.

3.2. Pencarian

Untuk melihat bagaimana GWO secara teoritis mampu menyelesaikan optimasi masalah, beberapa poin dapat dicatat:

a. Hirarki sosial yang diusulkan membantu GWO untuk mendapatkan solusi terbaik, yang diperoleh sejauh ini selama iterasi.
b. Mekanisme pengepungan yang diusulkan mendefinisikan lingkaran berbentuk lingkungan sekitar solusi yang bisa diperluas dimensi untuk memperoleh hasil yang lebih tinggi.
c. Metode berburu yang diusulkan memungkinkan solusi kandidat untuk cari posisi kemungkinan mangsa.
d. GWO hanya memiliki dua parameter utama yang harus disesuaikan *(a dan c)*

Untuk mengetahui algoritma GWO dapat dilihat dari *pseudocode* yang dijelaskan pada gambar 3.4.

```
Initialize the grey wolf position.
Initialize a,A and C
Compute the fitness of each search agents in the pack
Set the Xa,Xb,Xc according to the fitness
t=1
While(t<Max)
    fpr each wolf
        Update the position by equation(5)
    end for
    Update a,A and C
    Compute the fitness of each search agents in the
    pack
    Update the Xa,Xb,Xc
    for Xa,Xb,Xc
        Update the position by equation(13)
        Compute the fitness and update it accprding
    fitness
    t=t+1
end while
Output X
```

Gambar 3. 4 *Pseudocode* Algoritma GWO [1]

4. Algoritma BAT

4.1. Inisiasi

Bat Algorithm (BA) adalah algoritma metaheuristik yang tergolong baru, diciptakan oleh **Xin-She Yang** berdasarkan perilaku *echolocation* kelelawar. Menurut beliau, kemampuan *echolocation* dari microbat sangat mengagumkan karena kelelawar-kelelawar tersebut dapat menemukan mangsa mereka dan dapat membedakan beberapa jenis serangga yang berbeda, bahkan dalam keadaan gelap total. Formulasi algoritma ini dibuat dengan mengidealkan perilaku *echolocation* dari

kelelawar. Sebagai algoritma yang dikembangkan dengan mengkombinasikan keuntungan dari PSO, *genetic algorithm* dan juga *Harmony Search*, Bat *Algorithm* dianggap lebih unggul dibandingkan algoritmaalgoritma tersebut. Bat *Algorithm* diciptakan untuk menyelesaikan permasalahan optimasi fungsi kontinu, namun masih sedikit sekali riset yang menguji performansi Bat *Algorithm* dalam permasalahan tersebut jika dibandingkan dengan algoritma yang sudah ada sebelumnya [9].

4.2. Pencarian

Untuk proses pencarian pada algoritma BAT bisa dilihat dari *pseudocode* yang telah dijelaskan pada gambar 3.5.

```
Objective function f(x),x=[x1,x2,....,x4]^T
    Initialize the bat population xi(i=1,2,...,n) and vi
    Define pulse frequency fi at xi
    Initialize pulse rates ri and the loudness Ai
    While (t<Max number of iterations)
        Generate new solutions by adjusting
    frequency.
        and updating velocities and
    locations/solutions[(1)]
      if(rand>ri)
          Select a solutionamong the best solutions
          Generate a local solution around the
    selected best solution
      end if
          Generate a new solution by flying
    randomly
      if(rand<Ai&f(xi)<f(Xo))
          Accept the new solutions
          Increase ri and reduce Ai
      end if
        Rank the bats and find the current best Xo
    end while
          Postprocess result and visualization.
```

Gambar 3. 5 *Pseudocode* Algoritma BAT [4]

5. Algoritma CS

5.1. Inisiasi

Pada permulaan proses, matriks *fuzzy* untuk setiap *cuckoo* akan diinisialisasi secara random, dan dapat dituliskan dalam bentuk sebagai berikut.

$$R = (r_{ij})_{m \times n} = \begin{bmatrix} r_{11} & \cdots & r_{1n} \\ \vdots & \ddots & \vdots \\ r_{n1} & \cdots & r_{nn} \end{bmatrix}, r_{ij} \in [0, 1] \quad (3)$$

dimana, kondisi awal dari matriks yang diinisialisasi tersebut memenuhi syarat Rij [0, 1] dan jika i = j, Rij = 0. Setelah matriks selesai diinisialisasi, matriks kemudian dikonversi menjadi sebuah jalur kota menggunakan cara yang telah dijelaskan sebelumnya yaitu defuzzifikasi dengan metode angka terbesar (*Max Number Method*). Setelah proses defuzzifikasi selesai untuk semua *cuckoo*, proses dilanjutkan dengan menginisialisasi sarang yang terbaik, dengan melakukan seleksi berdasarkan nilai fitness diantara semua sarang *cuckoo* yang telah tersedia. Semakin kecil nilai fitness, maka semakin bagus sarang tersebut. Nilai *fitness* didapatkan dengan cara mencari jalur total yang menghubungkan antar tiap kota.

5.2. Pencarian

Proses pencarian terbagi ke dalam dua tahapan yaitu proses pencarian sarang dan proses penggantian sarang terburuk. Proses pencarian sarang merupakan tahapan di mana akan dilakukan pemilihan secara acak dari sejumlah *cuckoo* yang tersedia, di mana pada sarang yang ditinggalinya akan dilakukan random *step* dengan menggunakan *Lévy flights* sebagai acuan *step*, sedangkan proses penggantian sarang terburuk merupakan proses penggantian sejumlah sarang *cuckoo* yang dianggap buruk dengan persentase pergantian berdasarkan input(PA) yang diterima.

Untuk proses pencarian pada algoritma CS bisa dilihat dari *pseudocode* yang telah dijelaskan pada gambar 3.6.

```
begin
    Objective function f(x)
    Generate initial population of n host nest
    Evaluate fitness and rank eggs
    while(t>MaxGeneration) or Stop criterion
        t=t+1
        Get a cuckoo randomly/generate new
    solution by Levy flights
        Evaluate quality/fitness, Fi
        Choose a random nest j
        if(Fi>Fj)
          Replace j by the new soluton
        end if
         Worst nest is abandoned with probability
    Pa and new nest is built
          Evaluate fitness and Rank the solutions
    and find current best
    end while
        Post process result and visualization
end
```

Gambar 3. 6 *Pseudocode* Algoritma CS [5]

IV. TESTING

Pengujian dilakukan dengan menerapkan kelima metode algoritma *metaheuristic* untuk mencari nilai bobot yang paling optimal, nilai tersebut nantinya akan digunakan untuk melakukan klasifikasi terhadap data penyakit *diabetes*. Metode klasifikasi yang digunakan adalah *Neural Network Multilayer* peceptron.

Dataset yang digunakan adalah *open dataset* yang didapat dari UCI *Repository*. Jumlah data sebanyak 768 set yang akan dibagi menjadi 70% (537 set) untuk fase *training* dan 30% (231 set) untuk fase *testing*. Terdapat 8 atribut pada dataset untuk menentukan penyakit diabetes.

Pengujiaan dengan kelima metode dilakukan dengan kondisi yang sama yaitu dengan jumlah iterasi 50 dan populasi 50. Dilakukan 30 kali *running* untuk setiap algoritma *metaheuristic optimizer* dan akan diambil hasil yang terbaik, terburuk dan rata-rata dari setiap algoritma,

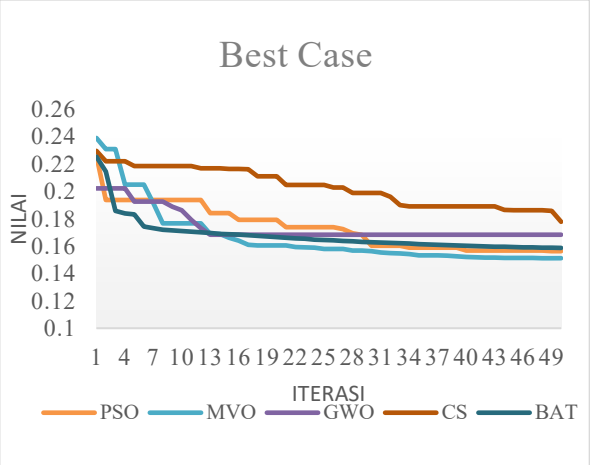
setelah itu dilakukan penghitungan akurasi *testing* terhadap 30% data *testing* dilakukan dengan 3 kali *cross-validation.*

V. HASIL PENGUJIAN

Dari pengujian, diperoleh grafik hasil perbandingan kelima algoritma *metaheuristic Optimization* (30x *Running*). Berikut adalah hasilnya:

A. Optimalisasi

a. *Best Case*



Gambar 5. 1 Grafik *Best Case*

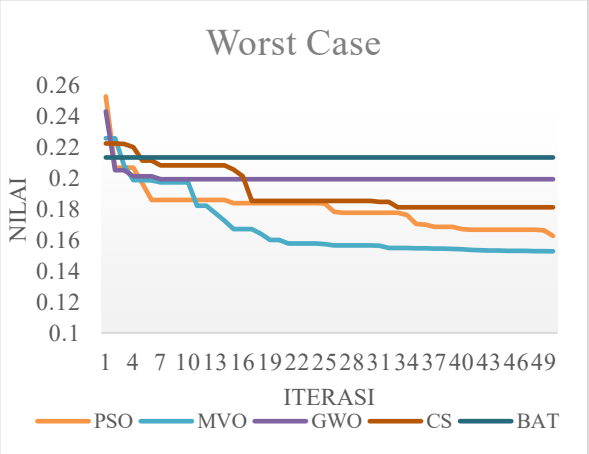
Dari grafik di atas, maka diperoleh hasil yang akan dijelaskan dari tabel 5.1.

***Tabel 5. 1 Hasil* Best Case**

Algoritma	Optimasi
PSO	0.156177
MVO	0.151077
GWO	0.168173
CS	0.177619
BAT	0.158556

Pada pengujiaan terbaik(*best case)* didapat bahwa algoritma *Multi-verse Optimization* (MVO) mendapat nilai yang paling baik yaitu **0.151077** dan algoritma CS mendapatkaan nilai paling buruk **0.177619.**

b. *Worst Case*



Gambar 5. 2 Grafik *Worst Case*

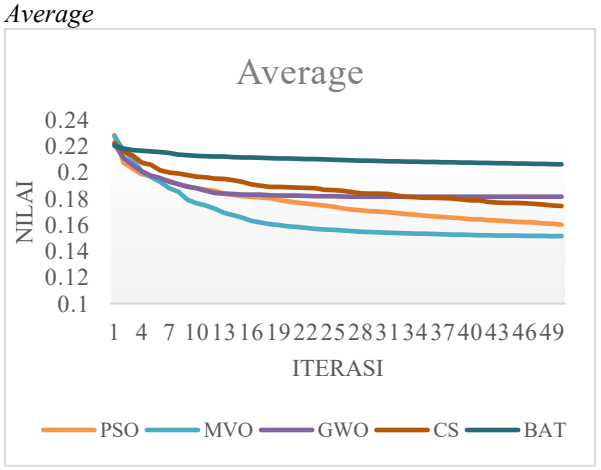
Dari grafik di atas, maka diperoleh hasil yang akan dijelaskan dalam tabel 5.2.

***Tabel 5. 2 Hasil* Worst Case**

Algoritma	Optimasi
PSO	0.162475
MVO	0.152604
GWO	0.199044
CS	0.180945
BAT	0.213043

Pada pengujiaan terburuk(*worst case)* didapat bahwa algoritma *Multi-verse Optimization* (MVO) mendapat nilai yang paling baik yaitu **0.152604** dan algoritma BAT mendapatkaan nilai paling buruk **0.213043.**

c. *Average*



Gambar 5. 3 Grafik *Average*

Dari grafik di atas, maka diperoleh hasil yang akan dijelaskan dalam tabel 5.3.

***Tabel 5. 3 Hasil* Average**

Algoritma	Optimasi
PSO	0.160069
MVO	0.151429
GWO	0.181448
CS	0.17432
BAT	0.206081

Pada pengujiaan rata-rata(*average case)* didapat bahwa algoritma *Multi-verse Optimization* (MVO) mendapat nilai yang paling baik yaitu **0.151429** dan algoritma BAT mendapatkaan nilai paling buruk **0.206081.**

B. Akurasi

Hasil Akurasi dan confusion matrix dari setiap algoritma (3X *cross-validation*). Berikut adalah hasil dari *best case* yang akan dijelaskan pada tabel 5.4., dan hasil *worst case* pada tabel 5.5.

a. *Best Case*

***Tabel 5. 4 Hasil* Best Case**

	Akurasi	Tpositif	Fnegatif	Fpositif	Tnegatif
PSO	78%	47	46	14	144
MVO	78%	48	45	16	142
GWO	68%	65	28	52	106
CS	76%	45	48	16	142
BAT	77%	47	46	18	140

b. *Worst Case*

Tabel 5. 5 Hasil *Worst Case*

	Akurasi	Tpositif	Fnegatif	Fpositif	Tnegatif
PSO	71%	29	64	14	144
MVO	75%	43	50	17	141
GWO	34%	91	2	158	0
CS	68%	29	64	21	137
BAT	33%	93	0	158	0

Penghitungan Standar Deviasi dan waktu eksekusi dari tiap algoritma (30X *Running*) dijelaskan pada tabel 5.6. berikut

Tabel 5. 6 Standar Deviasi

	Standar Deviasi	Waktu Eksekusi/Detik
PSO	0.01884	41.3
MVO	0.00675	39.1
GWO	0.12734	37.3
CS	0.02272	71.8
BAT	0.14571	39.6

VI. KESIMPULAN

Dari pengujian lima algoritma *metaheuristic optimization* untuk mencari bobot optimal pada neural *network classifier* untuk menentukan penderita diabetes dapat disimpulkan bahwa algoritma *Multi*-verse *Optimization* (MVO) memberikan hasil yang paling baik untuk kasus ini dengan memberikan akurasi pengujian paling tinggi dalam best case maupun *worst case*, serta nilai optimasi paling kecil diantara algoritma *metaheuristic* yang lain (semakin mendekati 0 semakin optimal). Algorima MVO juga memperoleh nilai stardar deviasi paling kecil walapun waktu eksekusi masih tertinggal dibandingkan dengan algoritma *Grey Wolf Optimization* (GWO).

IV. DAFTAR PUSTAKA

[1] S. Mirjalili, S. M. Mirjalili and A. Lewis, "Grey Wolf Optimizer," 2013.

[2] J. &. E. R. Kennedy, "Particle Swarm Optimization," 1995.

[3] S. Mirjalili, S. M. Mirjalili and A. Hatamlou, "Multi-Verse Optimizer: a nature-inspired algorithm for global optimization," 2015.

[4] X.-S. Yang, "A New Metaheuristic Bat-Inspired Algorithm," 2010.

[5] X.-S. Yang and S. Deb, "Cuckoo Search via Levy Flights," 2009.

[6] A. F. M, A. Fariza and I. Prasetyaningrum, "PENCARIAN JALUR TERPENDEK MENGGUNAKAN ALGORITMA PARTICLE SWARM OPTIMIZATION (PSO) DI KABUPATEN BANGKALAN".

[7] B. Santoso, Menu Sehat Alami untuk Batita dan Balita, Jakarta: Demedia, 2010.

[8] F. Sulistiowati, I. Cholissodin and &. Marji, Optimasi Penyusunan Bahan Makanan Sehat untuk Pemenuhan Gizi Keluarga dengan Algoritma Evolution Strategies, Malang: Universitas Brawijaya Malang, 2016.

[9] P. Aditya R and Suyanto, "ANALISIS DAN IMPLEMENTASI BAT ALGORITHM UNTUK CONTINUOUS OPTIMIZATION TASK," 2012.

[10] S. Mirjalili, "Moth-Flame Optimization Algorithm: A Novel Nature-inspired Heuristic Paradigm," 2015.