Available online at: http://journal.uny.ac.id/index.php/pythagoras
PYTHAGORAS: Jurnal Matematika dan Pendidikan Matematika, 17 (1), 2022, 266-280

Algoritme *Migrating Birds Optimization* dan Algoritme *Particle Swarm Optimization*: Penyelesaian Masalah *Knapsack* 0-1

Bib Paruhum Silalahi [1*], Mohamad Novanto [1], Prapto Tri Supriyo [1]
[1] Departemen Matematika, IPB University, Indonesia
* Corresponding Author. E-mail: bibparuhum@gmail.com

ARTICLE INFO

Article History:
Received: 10-Nov. 2020
Revised: 26-Jun. 2022
Accepted: 18-Sep. 2022

Keywords:
Masalah *knapsack*, metode *meta-heuristic*, *migrating birds optimization*, optimisasi, *particle swarm optimization*

ABSTRACT

Permasalahan *knapsack* merupakan salah satu masalah optimisasi. Masalah *knapsack* merupakan suatu permasalahan bagaimana memilih objek dari beberapa objek yang akan dimasukkan ke media penyimpanan dengan masing-masing objek memiliki bobot dan total bobot dari objek yang dipilih tidak boleh melebihi kapasitas media penyimpanannya, sehingga diperoleh nilai yang maksimal. Ketika objek yang dimasukkan ke dalam media penyimpanan bersifat harus dimasukkan semua atau tidak sama sekali, permasalahan ini dikenal dengan nama *knapsack* 0-1. Salah satu metode penyelesaian masalah *knapsack* 0-1 adalah dengan menggunakan metode *meta-heuristic*. Terdapat beberapa metode *meta-heuristic* seperti algoritme *Migrating Birds Optimization* dan *Particle Swarm Optimization*. Paper ini membahas bagaimana algoritme *Migrating Birds Optimization* dan *Particle Swarm Optimization* digunakan dalam menyelesaikan permasalahan *knapsack* 0-1. Kemudian dilakukan perbandingan kinerja kedua algoritme tersebut berdasarkan nilai fungsi tujuan untuk beberapa studi kasus. Berdasarkan hasil penelitian ini algoritme *Migrating Birds Optimization* mempunyai nilai hasil fungsi objektif yang lebih baik dibandingkan dengan algoritme *Particle Swarm Optimization*.

The knapsack problem is one of the optimization problems. The knapsack problem is a problem of how to select objects from several objects to be inserted into the storage with each object having a weight and the total weight of the selected object must not exceed the capacity of the storage so that the maximum value is obtained. When objects that are inserted into the storage have the character of having to be included all or nothing, this problem is known as the 0-1 knapsack. One of the methods of solving the 0-1 knapsack problem is by using the meta-heuristic method. There are several meta-heuristic methods such as the Migrating Birds Optimization algorithm and Particle Swarm Optimization. This paper discusses how Migrating Birds Optimization and Particle Swarm Optimization algorithms are used to solve the 0-1 knapsack problem. Then the performance of the two algorithms is compared based on the objective function values for several case studies. Based on the results of this study, the Migrating Birds Optimization algorithm has better objective function values than the Particle Swarm Optimization algorithm.

How to Cite:
Silalahi, B. P., Novanto, M., & Supriyo, P. T. (2022). Algoritme migrating birds optimization dan algoritme particle swarm optimization: penyelesaian masalah knapsack 0-1. *Pythagoras: Jurnal Matematika dan Pendidikan Matematika, 17*(1), 266-280. https://doi.org/10.21831/pythagoras.v17i1.35660

https://doi.org/10.21831/pythagoras.v17i1.35660

PENDAHULUAN

Optimisasi adalah suatu permasalahan matematika dalam mencari nilai yang terbaik dari suatu problem dengan memenuhi persyaratan-persyaratan yang ada. Beberapa penelitian tentang optimisasi seperti: menentukan pemilihan proyek pemasaran dengan *zero-one goal programming* (Silalahi et al., 2020), menentukan rute kendaraan dengan pengemudi sesekali (Lalang et al., 2018; Making et al., 2018) , menentukan frekuensi bus sebagai alat transportasi umum (Mayyani et al., 2017).

Permasalahan *knapsack* merupakan salah satu masalah optimisasi. Masalah *knapsack* merupakan suatu permasalahan bagaimana memilih objek dari beberapa objek yang akan dimasukkan ke media penyimpanan dengan masing-masing objek memiliki bobot dan total bobot dari objek yang dipilih tidak boleh melebihi kapasitas media penyimpanannya, sehingga diperoleh keuntungan yang maksimal (Kellerer et al., 2004). Ketika objek yang dimasukkan ke dalam media penyimpanan bersifat harus dimasukkan semua atau tidak sama sekali, permasalahan ini dikenal dengan nama knapsack 0-1.

Metode yang dapat digunakan untuk menyelesaikan permasalahan *knapsack* 0-1 salah satunya adalah metode *meta-heuristic*. Pada dasarnya, *meta-heuristic* adalah metode pendekatan yang dapat diterapkan untuk menyelesaikan di hampir semua masalah optimisasi. Metode ini dapat dipandang sebagai metode lanjut (*advanced*) berbasis heuristic untuk menyelesaikan permasalahan optimisasi secara efisien. Perbedaan utama antara metode *heuristic* dan *meta-heuristic* ialah metode *heuristic* bersifat problem dependent yang artinya bergantung pada permasalahan, jadi metode ini hanya dapat digunakan untuk jenis permasalahan tertentu, sedangkan metode *meta-heuristic* bersifat *problem independent* yang artinya tidak bergantung pada jenis permasalahan atau dapat digunakan untuk berbagai jenis permasalahan. Contoh metode *meta-heuristic* adalah *migrating birds optimization* (MBO) (Zhang et al., 2020), *ant colony optimization* (ACO) (Silalahi et al., 2019; Silalahi et al., 2020), *particle swarm optimization* (PSO) (Bisilisin et al., 2017), varian dari PSO (Zouache et al., 2018), algoritme genetika (Wihartiko et al., 2017), metode titik interior (Silalahi, 2019; Silalahi et al., 2019).

Beberapa peneliti telah menyelesaikan masalah *knapsack* 0-1 menggunakan metode *meta-heuristic* diantaranya menggunakan algoritme genetika (Singh, 2011), algoritme *global harmony search* (Zou et al., 2011), algoritme *shuffled frog leaping* (Bhattacharjee & Sarmah, 2014), algoritme *soccer league competition* (Iryanto & Mardiyati, 2015). Tujuan dari penelitian adalah mengetahui penyelesaian permasalahan *knapsack* 0-1 menggunakan algoritme MBO dan algoritme PSO, serta membandingkan kinerja MBO dan PSO dengan melihat nilai fungsi objektif yang diperoleh oleh kedua algoritme tersebut.

METODE

Penelitian penerapan algoritme *migration birds optimization (*MBO) dan algoritme *particle swarm optimization* (PSO) pada permasalahan *knapsack* 0-1 dilakukan dengan langkah-langkah berikut:

a. Mengidentifikasi data, dilakukan dengan memilih data yang sesuai dengan permasalahan *knapsack*.
b. Menerapkan algoritme MBO dan algoritme PSO pada data yang telah teridentifikasi.
c. Mengatur kriteria pemberhentian pada kedua algoritme tersebut, dipilih setelah 75 iterasi.
d. Membandingkan kedua algoritme menggunakan program yang telah dibuat berdasarkan hasil keuntungan maksimal.

Berikut ini adalah permasalahan *knapsack,* algoritme MBO dan algoritme PSO.

Permasalahan *knapsack*

Knapsack diartikan sebagai sebuah wadah atau tas. Permasalahan *knapsack* merupakan permasalahan tentang pemilihan barang dari sejumlah barang yang memiliki bobot atau nilai yang berbeda-beda, sehingga diperoleh keuntungan yang maksimal tanpa melebihi kapasitas dari wadah atau tas yang tersedia. Permasalahan *knapsack* dimisalkan dengan himpunan barang N, yang terdiri dari n objek i dengan keuntungan p_i, dan berat w_i, serta kapasitas wadah M (semua parameter adalah bilangan bulat positif) tujuannya untuk memilih subset dari N dengan total barang yang dipilih mempunyai keuntungan maksimal dan berat total tidak melebihi kapasitas yang ditentukan (Kellerer et al., 2004). Permasalahan *knapsack* yang akan dibahas adalah permasalahan *knapsack* 0-1. Permasalahan *knapsack* 0-1 memiliki sistem kerja dimana setiap objek harus diangkut seluruhnya atau tidak sama sekali. Permasalahan *knapsack* 0-1 memiliki solusi penyelesaian yang dinyatakan sebagai himpunan $X = \{x_1, x_2, x_3, \dots, x_n\}$, dimana x_i = 1 jika objek ke-i dimasukkan ke dalam media penyimpanan, dan $x_i = 0$ jika objek ke-i tidak dimasukkan ke dalam media penyimpanan.

Secara matematis, permasalahan *knapsack* 0-1 adalah sebagai berikut:

Misalkan:

Z : nilai optimum dari fungsi tujuan
z : kendala fungsi tujuan
n : banyak objek

p_i : keuntungan objek ke-i, dimana $i = 1,2,3,\ldots,n$
w_i: berat objek ke-i, dimana $i = 1,2,3,\ldots,n$
M : kapasitas media penyimpanan

Variabel keputusan: $x_i = \begin{cases} 1, & \text{jika objek ke-i dipilih} \\ 0, & \text{jika objek ke-i tidak dipilih} \end{cases}$

Fungsi objektif *knapsack* ialah memaksimumkan keuntungan dari barang yang dipilih, yaitu:

$$\max Z = \sum_{i=1}^{n} p_i x_i$$

Kendala *knapsack* ialah total bobot dari semua objek yang terpilih tidak boleh melebihi kapasitas media penyimpanan, yaitu

$$z = \sum_{i=1}^{n} w_i x_i \leq M$$

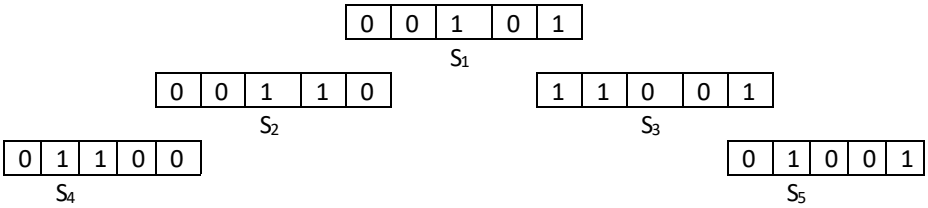
Algoritme *Migrating Birds Optimization* (MBO)

Algoritme MBO merupakan salah satu metode *meta-heuristic* yang terinspirasi oleh formasi huruf "V" yang dibentuk oleh kawanan burung ketika bermigrasi (Duman et al., 2012).

Pemecahan permasalahan optimisasi menggunakan algoritme MBO sebagai berikut:

a. Inisialisasi

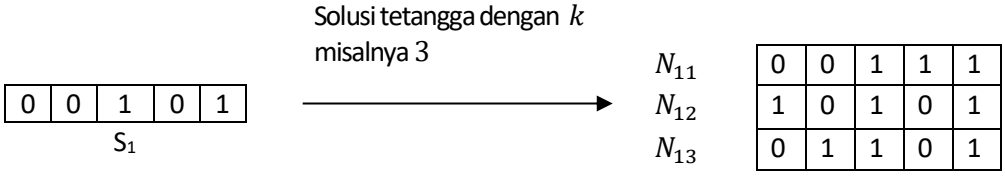
Penetapan populasi (n) pada algoritme MBO dilakukan secara bebas dengan syarat populasi tersebut adalah ganjil (sedikitnya 3) dan dapat dibentuk menjadi formasi V. Setiap individu dalam populasi merupakan solusi permutasi di ruang pencarian nilai optimal. Panjang permutasi sama dengan jumlah objek dan memiliki dua nilai yang berbeda (0 atau 1). Misalkan pada suatu kawanan terdiri dari 5 burung yang dilambangkan dengan S_i (i merupakan indeks burung) dan membentuk suatu formasi V sebagai berikut:



Panjang permutasi dari setiap burung adalah 5 yang berarti merepresentasikan banyaknya objek yang akan ditentukan apakah objek dipilih atau tidak dipilih. Nilai 1 pada solusi setiap burung merepresentasikan objek dipilih, dan nilai 0 objek tidak dipilih. Sebagai contoh, burung S_1 memiliki solusi 00101, berarti objek ke 3 dan 5 dipilih dan objek ke 1, 2, dan 4 tidak dipilih.

b. Solusi Tetangga dan Pembagian Solusi

Solusi pada setiap burung disebut solusi utama. Solusi tetangga setiap burung didapatkan dari evaluasi nilai solusi utama pada setiap burung. Evaluasi nilai solusi utama dilakukan dengan cara memilih secara acak salah satu sel yang bernilai 0 dan menggantinya dengan 1. Jika hasilnya melebihi kapasitas, maka salah satu sel dari permutasi yang bernilai 1 dipilih secara acak dan digantikan dengan nilai 0. Dengan cara ini, solusi-solusi tetangga didapatkan dan dilanjutkan dengan mengurutkan solusi-solusi tetangga dari nilai *fitness* terbesar ke nilai *fitness* terkecil. Solusi tetangga terbaik (yang memiliki *fitness* terbesar) akan digunakan untuk membandingkan dengan solusi utama, jika solusi utama lebih kecil dari solusi tetangga terbaik maka solusi tetangga tersebut akan menjadi solusi utama yang baru. Solusi tetangga yang tidak digunakan akan dibagikan ke burung berikutnya, yaitu burung lain yang berada di belakang barisan. Misalnya, jika jumlah solusi tetangga setiap burung sebanyak k dengan sisa solusi burung pemimpin sebanyak $k-1$ maka solusi tersebut dibagikan $\frac{k-1}{2}$ ke burung sebelah kiri dan $\frac{k-1}{2}$ ke burung sebelah kanan. Pembangkitan solusi tetangga pada burung pemimpin diilustrasikan sebagai berikut:



N_{ij} disimbolkan sebagai *neighbor* (solusi tetangga) pada S_i dengan tetangga ke $-j$. Evaluasi solusi utama dengan memilih secara acak nilai 0 dan digantikan dengan nilai 1 sehingga menghasilkan solusi tetangga sebanyak k. Misalnya, N_{11} merupakan solusi tetangga terbaik dan akan dibandingkan dengan solusi utama dari S_1. Jika nilai *fitness* N_{11} lebih besar dari nilai *fitness* S_1, maka solusi permutasi dari N_{11} akan menjadi solusi utama yang baru dari S_1. Jika nilai *fitness* N_{11} lebih kecil dari nilai *fitness* S_1, maka solusi utama dari S_1 tidak berubah dan solusi N_{11} akan dibuang. N_{12} dan N_{13} merupakan solusi tetangga yang tersisa (tidak digunakan untuk dibandingkan dengan solusi utama) akan dibagikan dengan burung berikutnya yang berada di sebelah kiri dan di sebelah kanan.

N_{11}	0	0	1	1	1
N_{12}	1	0	1	0	1
N_{13}	0	1	1	0	1

$S_2 \rightarrow N_{21}$

1	0	1	0	1

$S_3 \rightarrow N_{31}$

0	1	1	0	1

Karena $k = 3$ maka S_2 dan S_3 masih perlu 2 solusi tetangga lagi, pembangkitan solusi tetangga dilakukan dengan cara yang sama. Parameter untuk menjalankan algoritme MBO sebagai berikut:

$k \in \mathbb{Z}^+; k = \{3,5,7 \ldots\}, \quad x \in \mathbb{Z}^+; x = \{1,2, \ldots, \frac{(k-1)}{2}\}, \quad b = k - x$

Keterangan:

k = jumlah solusi tetangga setiap burung.

x = jumlah solusi tetangga yang dibagikan untuk solusi berikutnya.

b = jumlah solusi tetangga yang akan dibangkitkan selain burung pemimpin.

c. Kriteria pemberhentian

Proses iterasi akan terus berlangsung hingga kriteria pemberhentian terpenuhi. Kriteria pemberhentian pada algoritme MBO terpenuhi apabila iterasi yang dilakukan mencapai maksimal (Ulker & Tongur, 2017).

Algoritme *Particle Swarm Optimization* (PSO)

Particle swarm optimization merupakan algoritme berbasis populasi yang membahas individu dalam ruang pencarian nilai optimal. Algoritme ini didasarkan pada perilaku sosial dari sekawanan burung atau sekelompok ikan. Perilaku sosial tersebut terdiri dari tindakan setiap individu dalam suatu populasi dan pengaruh dari individu-individu lain dalam populasi. Contoh perilaku sosial dalam sekawanan burung adalah jika seekor burung menemukan jalan yang tepat atau pendek untuk menuju ke sumber makanan, burung lain dalam satu populasi akan mengikuti jalan tersebut meskipun lokasi mereka jauh. Dalam PSO, populasi burung tersebut disebut kawanan (*swarm*) dan individu burung disebut partikel. Setiap partikel berpindah dengan kecepatan yang diadaptasi dari daerah pencarian dan selalu menyimpan posisi terbaik yang pernah dicapai. Posisi partikel ke-i pada suatu kawanan dilambangkan dengan x_i, sedangkan posisi terbaik dari perjalanan suatu partikel disebut $Pbest_i$. Masing-masing partikel akan mengetahui posisi terbaik secara global yang ditemukan oleh salah satu anggota populasi, posisi terbaik dari seluruh partikel dalam kawanan disebut $Gbest$ (Liang et al., 2010).

Pada algoritme PSO, pencarian solusi dilakukan oleh suatu populasi yang terdiri dari beberapa partikel. Setiap partikel merepresentasikan posisi atau solusi dari permasalahan yang dihadapi. Pencarian solusi yang optimal dilakukan dengan cara setiap partikel menentukan posisi terbaik dari partikel tersebut (*local best*) dan menentukan posisi partikel terbaik dari kawanan (*global best*).

Pemecahan permasalahan optimisasi menggunakan algoritme PSO sebagai berikut:

a. Inisialisasi

Penetapan posisi setiap partikel dalam kawanan dilakukan secara acak dalam bentuk bilangan riil kemudian dikonversikan menjadi bilangan biner tanpa melebihi kapasitas *knapsack* yang telah ditentukan, apabila melebihi kapasitas maka akan diberlakukan *penalty*. Proses *penalty* dilakukan dengan merubah solusi posisi awal menjadi solusi yang tidak melebihi kapasitas dengan mengevaluasi objek yang bernilai 1 menjadi 0. Kemudian, penetapan kecepatan awal setiap partikel dilakukan secara acak pada suatu rentang tertentu. Kecepatan awal setiap partikel dibangkitkan dengan rumus $v_i = v_{min} + rand(0,1)(v_{max} - v_{min})$, dengan v_{min} merupakan kecepatan minimum dan v_{max} merupakan kecepatan maksimum

(Liang *et al*., 2010). v_{max} dan v_{min} ditentukan dengan rumus $v_{max} = \varepsilon(x_{max} - x_{min}), \varepsilon \in (0,1]$ dan $v_{min} = -v_{max}$. Misalkan pada suatu populasi terdiri dari 5 partikel yang dilambangkan dengan x_i (i merupakan indeks partikel) dan mempunyai nilai dan solusi awal sebagai berikut:

x_1	0.746	0.364	0.443	0.111	0.842
x_2	0.950	0.991	0.238	0.638	0.503
x_3	0.501	0.137	0.575	0.556	0.116
x_4	0.131	0.619	0.908	0.234	0.885
x_5	0.303	0.532	0.474	0.945	0.018

→

x_1	1	0	0	0	1
x_2	1	1	0	1	1
x_3	1	0	1	1	0
x_4	0	1	1	0	1
x_5	0	1	0	1	0

Apabila ada posisi dari partikel ke-i yang jika dievaluasi berdasarkan bobot objeknya melebihi kapasitas yang ditentukan, maka salah satu sel dari permutasi yang bernilai 1 dipilih secara acak dan digantikan dengan nilai 0. Misalnya partikel x_2 adalah solusi yang bobotnya melebihi kapasitas, sehingga partikel x_2 akan dievaluasi kembali sehingga mempunyai solusi yang bobotnya tidak melebihi kapasitas.

x_2	1	1	0	1	1	→	x_2	1	1	0	0	1

Kemudian, untuk membangkitkan kecepatan awal diperlukan v_{min} dan v_{max} . Misalkan, $v_{min} = -1$ dan $v_{max} = 1$, kecepatan awal dapat dihitung sebagai $v_i = (-1) + rand(0,1) * \big(1 — (-1)\big)$. Kecepatan untuk setiap partikel didapatkan sebagai berikut:

v_1	0.388	-0.860	0.285	0.924	-0.798
v_2	0.621	0.057	-0.853	-0.724	-0.261
v_3	-0.938	0.486	0.431	0.703	-0.375
v_4	-0.714	0.782	0.877	0.031	-0.713
v_5	-0.096	-0.306	-0.632	-0.838	-0.056

b. Perhitungan Setiap Partikel

Hitung *fitness* partikel dari solusi posisi awal yang sudah didapatkan secara acak dan tentukan $Pbest$-nya ($Pbest$ awal dari setiap partikel sama dengan solusi posisi awal). Selanjutnya, cari nilai *fitness* terbesar dari $Pbest$ semua partikel untuk menentukan $Gbest$-nya. Misalkan, posisi awal masing-masing partikel mempunyai *fitness* dengan $Fitness_i = \{15,23,17,14,20\}$, maka nilai $x_i, Fitness_i, Pbest_i$, dan $Gbest$ dapat diilustrasikan sebagai berikut:

	Permutasi	*Fitness*	*Pbest*	*Gbest*
x_1	10001	15	10001	
x_2	11001	23	11001	
x_3	10110	17	10110	11001
x_4	01101	14	01101	
x_5	01010	20	01010	

Kemudian, untuk menentukan posisi dan kecepatan yang baru, nilai yang digunakan adalah nilai yang belum diubah menjadi bilangan biner. Nilai kecepatan dan posisinya diperbarui dengan persamaan berikut (Chih et al., 2014):

$$v_i(t+1) = wv_i(t) + c_1r_1\big(Pbest_i - x_i(t)\big) + c_2r_2\big(Gbest_t - x_i(t)\big),$$
$$x_i(t+1) = x_i(t) + v_i(t+1)$$

dengan:

$v_i(t)$: kecepatan partikel ke i pada iterasi ke t,

$v_i(t+1)$: kecepatan partikel ke i setelah diperbarui pada iterasi ke- $(t+1)$,

$x_i(t)$: posisi partikel ke i pada iterasi ke t,
$x_i(t+1)$: posisi partikel ke i setelah diperbarui pada iterasi ke- $(t+1)$,
c_1, c_2 : konstanta percepatan,
r_1, r_2 : dua variabel acak terdistribusi *uniform* antar 0 sampai 1,
w : bobot inersia yang menunjukkan pengaruh perubahan kecepatan,
$Pbest_i$: posisi terbaik dari partikel ke-i,
$Gbest_t$: posisi partikel terbaik dari seluruh partikel dalam kawanan pada iterasi ke t.

Nilai x_i yang baru perlu diubah menjadi bilangan biner untuk dihitung berat dan *fitness* setiap partikelnya. Apabila total berat dari posisi partikel yang baru melebihi kapasitas, maka akan diberlakukan *penalty*. Proses *penalty* dilakukan dengan merubah solusi posisi awal menjadi solusi yang tidak melebihi kapasitas dengan mengevaluasi objek yang bernilai 1 menjadi 0. Setelah itu, hitung *fitness* setiap partikel dan tentukan $Pbest$-nya. Jika *fitness* partikel ke-i yang baru lebih besar dibandingkan *fitness* $Pbest$-nya, maka $Pbest$ akan digantikan dengan solusi partikel yang baru. Cari nilai *fitness* terbesar dari $Pbest$ semua partikel untuk menentukan $Gbest$ yang baru. Jika nilai *fitness* maksimum dari $Pbest_i$ lebih besar dari nilai *fitness* $Gbest$ sebelumnya, maka $Gbest$ akan digantikan dengan solusi permutasi dari $Pbest$ maksimum tersebut.

	Permutasi lama	*Fitness* lama	Permutasi baru	*Fitness* baru	*Pbest* baru	*Fitness Pbest*	*Gbest* baru
x_1	10001	15	01101	16	01101	16	
x_2	11001	23	10101	19	11001	23	
x_3	10110	17	00111	25	00111	25	00111
x_4	01101	16	11100	22	11100	22	
x_5	01010	20	10110	17	01010	20	

Jika perhitungan satu iterasi dilakukan, maka solusi yang didapatkan adalah $Gbest$ baru dengan solusi permutasinya 00111 yang artinya objek ke-1 dan 2 tidak dipilih, objek ke-3, 4 dan 5 akan dipilih dengan menghasilkan *fitness* sebesar 25. Model ini akan disimulasikan dalam sejumlah iterasi, sehingga disetiap iterasi posisi partikel akan semakin mengarah terhadap fungsi tujuannya.

c. Kriteria pemberhentian

Proses iterasi akan terus berlangsung hingga kriteria pemberhentian terpenuhi. Kriteria pemberhentian pada algoritme PSO akan terpenuhi apabila iterasi yang dilakukan mencapai maksimal.

HASIL PENELITIAN

Pada penelitian ini, dilakukan penyelesaian terhadap tiga kasus *knapsack* menggunakan algoritme MBO dan PSO. Hasil dari kedua metode akan dibandingkan satu sama lain pada keuntungan terbaik yang diperoleh. Data yang digunakan adalah data sejumlah barang yang akan dibeli oleh Toko X. Data mentah diambil dari Toko X berupa data harga beli barang, harga jual barang, dan banyaknya barang yang dibeli (Tabel 1).

Kemudian data pada Tabel 1 dibuat menjadi tiga kasus, kasus pertama mengambil data 5 barang pertama pada Tabel 1 dengan kapasitas 80 kg, kasus kedua mengambil data 28 barang pertama pada Tabel 1 dengan kapasitas 200 kg, kasus ketiga mengambil data seluruh barang pada Tabel 1 dengan kapasitas 270 kg. Data pada setiap kasus akan dilakukan praproses untuk mencari keuntungan (p_i) dan berat (w_i) dari masing-masing barang. Praproses untuk mencari p_i dilakukan dengan cara menghitung keuntungan masing-masing barang yang diperoleh dari selisih harga beli dan harga jual yang ditetapkan lalu dikalikan dengan banyaknya barang, sedangkan untuk mencari w_i dilakukan dengan cara mengalikan berat dari tiap barang dengan banyaknya barang. Data praproses disajikan pada Tabel 2.

Tabel 1. Data barang yang ingin dibeli toko X

No.	Nama Barang	Banyak Barang (item)	Harga Beli (Rp)	Harga Jual (Rp)	No.	Nama Barang	Banyak Barang (item)	Harga Beli (Rp)	Harga Jual (Rp)
1	Furadan 2 kg	10	19500	30500	21	Crowen 113 EC 80 g	50	13000	20000
2	Amegrass 10 kg	4	550000	675000	22	Crowen 113 EC 200 g	40	23500	30000
3	Polaram 1 kg	16	60000	68000	23	Crowen 113 EC 400 g	20	40000	49500
4	Plastik mulsa 18 kg	2	730000	790000	24	Dithane 200 g	15	17000	24000
5	Topsin M 500 g	20	62000	75000	25	Antracol 250 g	18	30500	35000
6	Klopindo 100 g	10	15000	23000	26	Antracol 500 g	12	52000	60000
7	Lannate Biru 100 g	10	25000	29000	27	Benstar 200 g	22	18000	23000
8	Lannate Merah 100 g	10	20000	27000	28	Masalgin 200 g	14	16000	20000
9	Lannate Merah 15 g	40	15000	25000	29	Phycozan 200 g	17	16000	21000
10	Furadan 3 gr 1kg	10	16000	24500	30	Acrobat 10 g	20	8000	12000
11	Metindo 25 wp 100 g	20	27500	35000	31	Acrobat 40 g	6	30000	37000
12	Confidor 100 g	10	26000	31500	32	Gandsi. D 500 g	10	22500	28000
13	Dangke 100 g	50	15000	20500	33	Gandsi. B 500 g	15	21000	30500
14	Dangke 40 WP 250 gr	20	42000	55000	34	Ridomild gold 100 g	50	24500	30000
15	Plastik mulsa 9 kg	5	305000	350000	35	Saromyl 25 g	20	27500	35000
16	Plastik mulsa 15 kg	4	457000	500000	36	Saromyl 5 g	50	8500	12000
17	Amegrass 5 kg	10	315000	350000	37	Procure 400 g	17	41000	50000
18	Teku 100 EC 100 g	50	21000	30000	38	Starmyl 100 g	32	48000	59000
19	Teku 100 EC 200 g	40	38000	45000	39	Dense 200 g	12	30000	38000
20	Teku 100 EC 400 g	20	72500	80000	40	BM Toplas 100 g	20	14500	20000
					41	BM Zebco 1 kg	5	52000	64000

Tabel 2. Data praproses masing-masing kasus

Kasus I		Kasus II				Kasus III			
w (kg)	p (Rp)	w (kg)	p (Rp)	w (kg)	p (Rp)	w (kg)	p (Rp)	w (kg)	p (Rp)
20	110000	20	110000	45	225000	20	110000	4	350000
40	500000	40	500000	60	172000	40	500000	8	260000
16	128000	16	128000	50	350000	16	128000	8	190000
36	120000	36	120000	5	450000	36	120000	3	105000
10	260000	10	260000	8	280000	10	260000	4.5	81000
		1	80000	8	150000	1	80000	6	96000
		1	40000	4	350000	1	40000	4.4	110000
		1	70000	8	260000	1	70000	2.8	56000
		0.6	400000	8	190000	0.6	400000	3.4	85000
		10	85000	3	105000	10	85000	0.2	80000
		2	150000	4.5	81000	2	150000	0.24	42000
		1	55000	6	96000	1	55000	5	55000
		5	275000	4.4	110000	5	275000	7.5	142500
		5	260000	2.8	56000	5	260000	5	275000
						45	225000	0.5	150000
						60	172000	0.25	175000
						50	350000	6.8	153000
						5	450000	3.2	352000
						8	280000	2.4	96000
						8	150000	2	110000
								5	60000

Penyelesaian Menggunakan Algoritme MBO

Pada penelitian ini, digunakan *software* MATLAB untuk mencari penyelesaian *knapsack* menggunakan algoritme MBO, menggunakan parameter yang disajikan pada Tabel 3.

Tabel 3. Parameter MBO yang digunakan untuk ketiga kasus

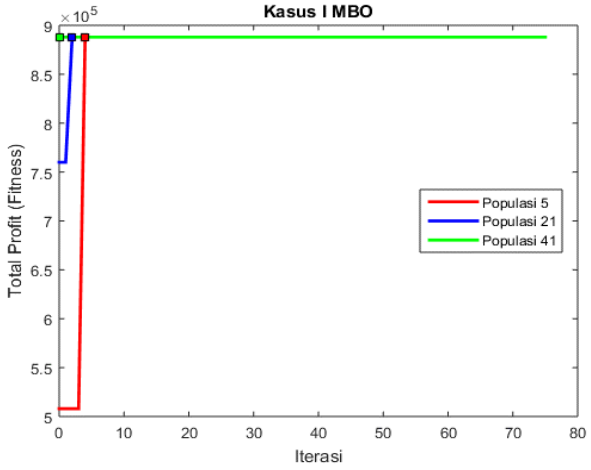
Parameter	Nilai
Jumlah burung (n)	5, 21, dan 41
Tetangga (k)	3
Tetangga yang dibagikan (x)	1
Maksimal iterasi	75

Kasus I

Pada kasus I dilakukan *running* program sebanyak 10 kali dan diperoleh hasil seperti pada Tabel 4. Selanjutnya akan dipilih nilai *fitness* terbesar untuk dijadikan sebagai solusi, karena dalam kasus I pada 3 populasi MBO nilai *fitness* terbesarnya sama, maka nilai *fitness* sebesar 888000 merupakan keuntungan yang didapatkan dari kasus I. Pada setiap populasi dengan nilai *fitness* terbesar (diambil pada hasil *running* nomor 2) akan dilihat laju per-iterasinya untuk mengetahui pada iterasi ke berapa nilai *fitness* mencapai nilai terbaik. Grafik nilai *fitness* pada setiap iterasi dapat dilihat pada Gambar 1. Pada populasi 5, nilai *fitness* mencapai nilai terbaik di iterasi ke-5. Pada populasi 21, nilai *fitness* mencapai nilai terbaik di iterasi ke-3. Pada populasi 41, nilai *fitness* mencapai nilai terbaik di iterasi ke-0.

Tabel 4. Hasil 10 kali running kasus I MBO

No.	Populasi 5		Populasi 21		Populasi 41	
	Fitness	Waktu	Fitness	Waktu	Fitness	Waktu
1	888000	0.47482	888000	0.57189	888000	0.67475
2	888000	0.38733	888000	0.58024	888000	0.69763
3	888000	0.48057	888000	0.59153	888000	0.68910
4	888000	0.40183	888000	0.57224	888000	0.69679
5	888000	0.48337	888000	0.59711	888000	0.68122
6	888000	0.50867	888000	0.56488	888000	0.69001
7	888000	0.48044	888000	0.57275	888000	0.73720
8	888000	0.51050	888000	0.57038	888000	0.68214
9	888000	0.48850	888000	0.59199	888000	0.70374
10	888000	0.49591	888000	0.58621	888000	0.67322
Rata-rata	888000	0.47119	888000	0.57992	888000	0.69258



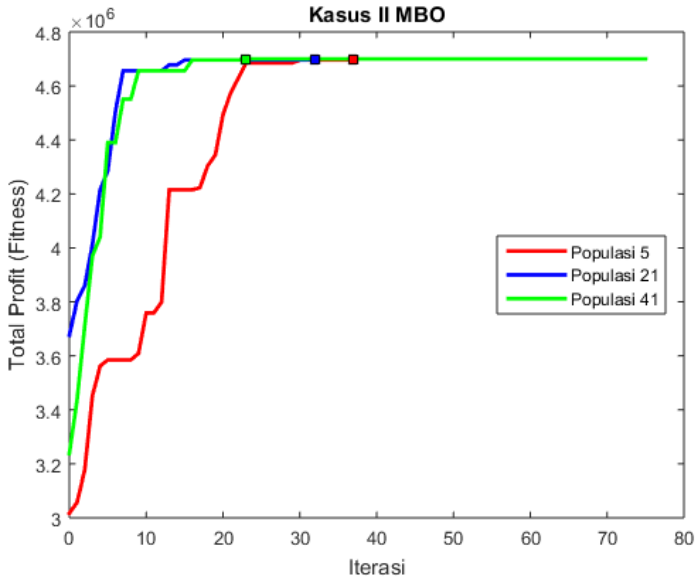
Gambar 1. Grafik nilai fitness kasus I MBO

Kasus II

Pada kasus II dilakukan *running* program sebanyak 10 kali dan diperoleh hasil seperti pada Tabel 5. Selanjutnya akan dipilih nilai *fitness* terbesar untuk dijadikan sebagai solusi, karena dalam kasus II pada 3 populasi MBO nilai *fitness* terbesarnya sama, maka nilai *fitness* sebesar 4700000 merupakan keuntungan yang didapatkan dari kasus II. Pada populasi dengan nilai *fitness* terbesar (diambil hasil *running* nomor 6) akan dilihat laju per-iterasinya. Grafik nilai *fitness* pada setiap iterasi dapat dilihat pada Gambar 2. Pada populasi 5, nilai *fitness* mencapai nilai terbaik di iterasi ke-38. Pada populasi 21, nilai *fitness* mencapai nilai terbaik di iterasi ke-33. Pada populasi 41, nilai *fitness* mencapai nilai terbaik di iterasi ke-24.

Tabel 5. Hasil 10 kali *running* kasus II MBO

No.	Populasi 5		Populasi 21		Populasi 41	
	Fitness	Waktu	*Fitness*	Waktu	*Fitness*	Waktu
1	4700000	0.83879	4700000	1.13963	4700000	1.46220
2	4700000	0.87361	4700000	1.18332	4700000	1.44693
3	4700000	0.86608	4700000	1.11417	4700000	1.37620
4	4700000	0.92122	4700000	1.16209	4700000	1.07781
5	4700000	0.85068	4700000	1.12670	4700000	1.49214
6	4700000	0.69765	4700000	1.00578	4700000	1.46512
7	4700000	0.86886	4700000	1.15796	4700000	1.62720
8	4700000	0.93097	4700000	1.23782	4700000	1.53850
9	4700000	0.94042	4700000	1.06141	4700000	1.62684
10	4700000	0.89461	4700000	1.09364	4700000	1.54105
Rata-rata	4700000	0.86829	4700000	1.12825	4700000	1.46539



Gambar 2. Grafik nilai *fitness* kasus II MBO

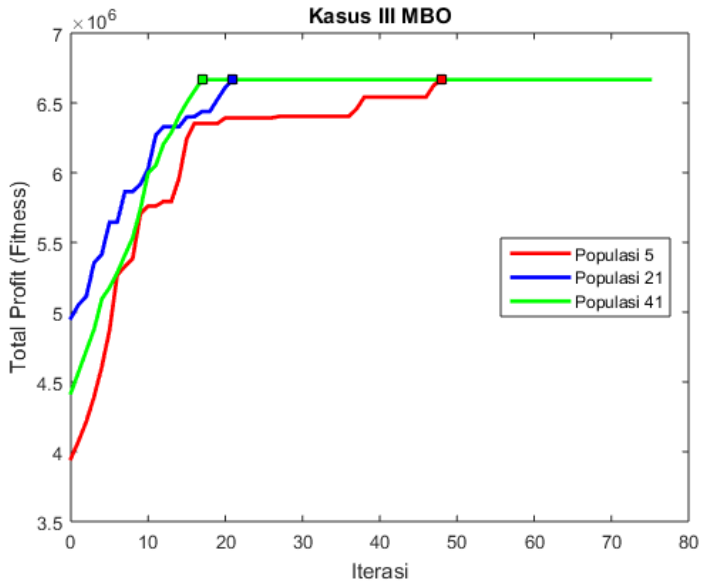
Kasus III

Pada kasus III dilakukan *running* program sebanyak 10 kali dan diperoleh hasil seperti pada Tabel 6. Selanjutnya akan dipilih nilai *fitness* terbesar untuk dijadikan sebagai solusi, karena dalam kasus III pada 3 populasi MBO nilai *fitness* terbesarnya sama, maka nilai *fitness* sebesar 6666500 merupakan keuntungan yang didapatkan dari kasus III. Pada setiap populasi dengan nilai *fitness* terbesar (diambil pada percobaan nomor 1) akan dilihat laju per-iterasinya untuk mengetahui di iterasi berapa nilai *fitness* mencapai nilai terbaik. Grafik nilai *fitness* pada setiap iterasi dapat dilihat pada Gambar 3. Pada populasi 5, nilai *fitness* mencapai nilai terbaik di iterasi ke-38. Pada

populasi 21, nilai *fitness* mencapai nilai terbaik di iterasi ke-33. Pada populasi 41, nilai *fitness* mencapai nilai terbaik di iterasi ke-24. Solusi permutasi MBO dari *fitness* terbaik untuk masing-masing kasus ditampilkan pada Tabel 7.

Tabel 6. Hasil 10 kali *running* kasus II MBO

No.	Populasi 5		Populasi 21		Populasi 41	
	Fitness	Waktu	Fitness	Waktu	Fitness	Waktu
1	6666500	1.01390	6666500	1.37518	6666500	1.65842
2	6593500	1.05530	6666500	1.34895	6666500	1.71703
3	6598500	1.08320	6666500	1.32662	6666500	1.68566
4	6666500	1.10520	6666500	1.42565	6666500	1.67392
5	6598500	1.13190	6666500	1.41506	6666500	1.68391
6	6568500	1.10440	6666500	1.43778	6666500	1.73828
7	6593500	1.14710	6666500	1.38960	6666500	1.72940
8	6666500	1.12460	6666500	1.27004	6666500	1.67836
9	6536500	1.09340	6666500	1.30442	6666500	1.61885
10	6545500	1.08630	6666500	1.33933	6666500	1.69982
Rata-rata	6603400	1.09453	6666500	1.36326	6666500	1.68836



Gambar 3. Grafik nilai *fitness* kasus III MBO

Tabel 7. Solusi permutasi MBO dari *fitness* terbaik

Jenis Kasus	Solusi	Hasil
Kasus I	01101	888000
Kasus II	01101111111111001111111110111	4700000
Kasus III	11101111111111001111111111111111111111111	6666500

Penyelesaian Menggunakan Algoritme PSO

Software MATLAB 2015a digunakan untuk mencari penyelesaian *knapsack* menggunakan algoritme PSO. Percobaan dilakukan menggunakan parameter yang disajikan pada Tabel 8 .

Kasus I

Pada kasus I dilakukan *running* program sebanyak 10 kali dan diperoleh hasil seperti pada Tabel 9. Selanjutnya akan dipilih nilai *fitness* terbesar untuk dijadikan sebagai solusi. Karena dalam kasus I pada 3 populasi

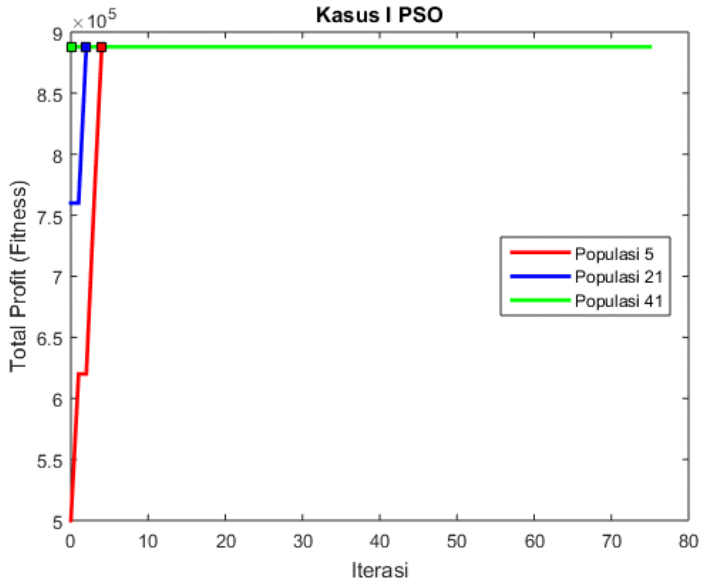
PSO nilai *fitness* terbesarnya sama, maka nilai *fitness* sebesar 888000 merupakan keuntungan yang didapatkan dari kasus I. Pada setiap populasi dengan nilai *fitness* terbesar (diambil pada percobaan nomor 3) akan dilihat laju per-iterasinya untuk mengetahui di iterasi berapa nilai *fitness* mencapai nilai terbaik. Grafik nilai *fitness* pada setiap iterasi dapat dilihat pada Gambar 4. Pada populasi 5, nilai *fitness* mencapai nilai terbaik di iterasi ke-4. Pada populasi 21, nilai *fitness* mencapai nilai terbaik di iterasi ke-2. Pada populasi 41, nilai *fitness* mencapai nilai terbaik di iterasi ke-0.

Tabel 8. Parameter PSO yang digunakan untuk ketiga kasus

Parameter	Nilai
Jumlah partikel (n)	5, 21, dan 41
Bobot inersia (w)	0.1
Konstanta percepatan (c_1 dan c_2)	2
Maksimal iterasi	75

Tabel 9. Hasil 10 kali *running* kasus I PSO

No.	Populasi 5		Populasi 21		Populasi 41	
	Fitness	Waktu	Fitness	Waktu	Fitness	Waktu
1	888000	0.41807	888000	0.49548	888000	0.53713
2	870000	0.40748	888000	0.50191	888000	0.59388
3	888000	0.40553	888000	0.49400	888000	0.51910
4	888000	0.49284	870000	0.50206	888000	0.53915
5	870000	0.41879	888000	0.48610	888000	0.52039
6	738000	0.40029	888000	0.48464	888000	0.52075
7	888000	0.43947	888000	0.48846	888000	0.51194
8	888000	0.45670	888000	0.49700	888000	0.53808
9	738000	0.44290	870000	0.51752	888000	0.54211
10	870000	0.49131	888000	0.50873	888000	0.52143
Rata-rata	852600	0.43734	884400	0.49759	888000	0.53439



Gambar 4. Grafik nilai *fitness* kasus I PSO

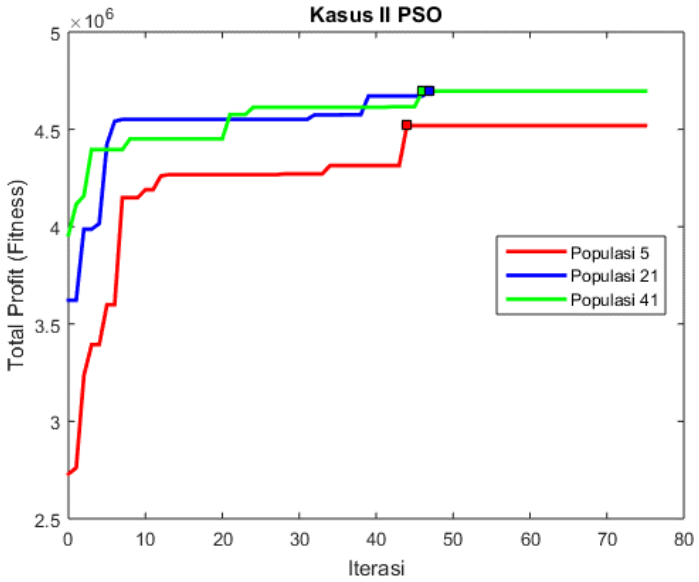
Kasus II

Pada kasus II dilakukan *running* program sebanyak 10 kali dan diperoleh hasil seperti pada Tabel 10. Selanjutnya akan dipilih nilai *fitness* terbesar untuk dijadikan sebagai solusi. Pada populasi 5 nilai *fitness* terbesarnya

adalah 4519000, populasi 21 nilai *fitness* terbesarnya adalah 4696000, populasi 41 nilai *fitness* terbesarnya adalah 4696000, maka nilai *fitness* sebesar 4696000 merupakan keuntungan yang didapatkan dari kasus II pada algoritme PSO dengan percobaan menggunakan populasi 5, 21, dan 41. Grafik nilai *fitness* pada setiap iterasi dapat dilihat pada Gambar 5. Pada populasi 5, nilai *fitness* mencapai nilai terbaik di iterasi ke-45. Pada populasi 21, nilai *fitness* mencapai nilai terbaik di iterasi ke-48. Pada populasi 41, nilai *fitness* mencapai nilai terbaik di iterasi ke-47.

Tabel 10. Hasil 10 kali *running* kasus II PSO

No.	Populasi 5		Populasi 21		Populasi 41	
	Fitness	Waktu	Fitness	Waktu	Fitness	Waktu
1	4235000	0.52710	4623000	0.70278	4605000	0.83433
2	4265000	0.62466	4505000	0.61906	4678000	0.73570
3	4360000	0.61041	4696000	0.68184	4696000	0.68405
4	4340000	0.58979	4685000	0.65551	4678000	0.91672
5	4519000	0.59744	4574000	0.64461	4631000	1.02380
6	4503000	0.60893	4510000	0.67251	4671000	1.01430
7	4383000	0.64778	4678000	0.67989	4608000	1.03270
8	4316000	0.62752	4575000	0.65826	4696000	0.95938
9	4436000	0.59307	4537000	0.96256	4645000	1.05670
10	4463000	0.59108	4597000	0.70579	4671000	0.82598
Rata-rata	4382000	0.60178	4598000	0.69828	4657900	0.90836



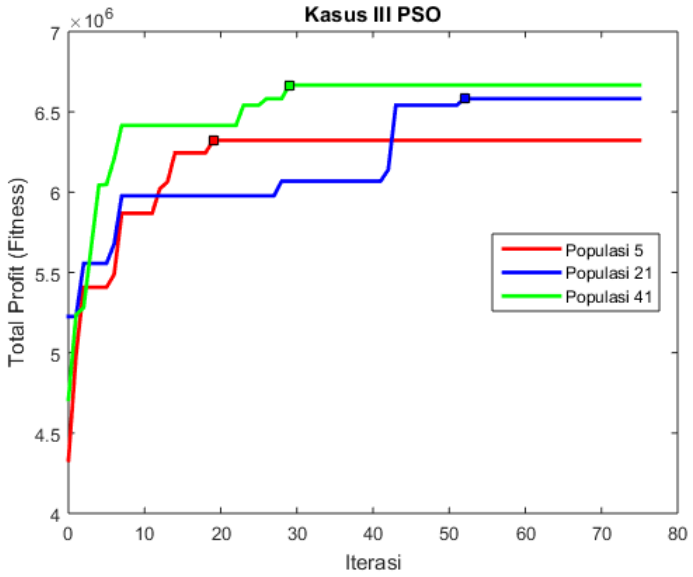
Gambar 5. Grafik nilai *fitness* kasus II PSO

Kasus III

Pada kasus III dilakukan *running* program sebanyak 10 kali dan diperoleh hasil seperti pada Tabel 11. Selanjutnya akan dipilih nilai *fitness* terbesar untuk dijadikan sebagai solusi. Pada populasi 5 nilai *fitness* terbesarnya adalah 6322500, populasi 21 nilai *fitness* terbesarnya adalah 6581500, populasi 41 nilai *fitness* terbesarnya adalah 6666500, maka nilai *fitness* sebesar 6666500 merupakan keuntungan yang didapatkan dari kasus III pada algoritme PSO dengan percobaan menggunakan populasi 5, 21, dan 41. Grafik nilai *fitness* pada setiap iterasi dapat dilihat pada Gambar 6. Pada populasi 5, nilai *fitness* mencapai nilai terbaik di iterasi ke-20. Pada populasi 21, nilai *fitness* mencapai nilai terbaik di iterasi ke-53. Pada populasi 41, nilai *fitness* mencapai nilai terbaik di iterasi ke-30. Solusi permutasi PSO dari *fitness* terbaik untuk masing-masing kasus ditampilkan pada Tabel 12.

Tabel 11. Hasil 10 kali *running* kasus II PSO

No.	Populasi 5		Populasi 21		Populasi 41	
	Fitness	Waktu	Fitness	Waktu	Fitness	Waktu
1	6295500	0.89826	6392500	1.03150	6541500	1.27140
2	6261000	0.87127	6318500	1.13340	6666500	1.13530
3	6181000	0.97180	6242500	0.99463	6611500	1.03070
4	6291500	0.95674	6376000	1.07600	6462500	1.26383
5	6047500	0.97180	6558500	1.09880	6548500	1.02550
6	6322500	0.94835	6581500	1.13120	6610500	1.13050
7	5941500	1.01630	6396500	1.08800	6502500	0.99482
8	6297500	0.99550	6445500	1.14490	6581500	0.91382
9	5880500	1.04450	6363500	1.11430	6611500	1.21658
10	6177500	1.02630	6505500	1.08920	6624500	1.33106
Rata-rata	6169600	0.97008	6418050	1.09019	6576100	1.13135



Gambar 6. Grafik nilai *fitness* kasus III PSO

Tabel 12. Solusi permutasi PSO dari *fitness* terbaik

Jenis Kasus	Solusi	Hasil
Kasus I	01101	888000
Kasus II	01101111101111001111111111111	4696000
Kasus III	11101111111111001111111111111111111111111	6666500

Perbandingan Hasil

Tabel 13 memperlihatkan total keuntungan terbaik yang didapatkan oleh masing-masing algoritme di setiap kasus. Pada kasus I, algoritme MBO dan PSO di setiap populasi yang dicobakan menghasilkan nilai *fitness* terbaik yang sama. Pada kasus II, hanya nilai *fitness* MBO yang memiliki kesamaan nilai pada setiap populasi yang dicobakan dan nilai *fitness* PSO memiliki selisih 4% pada populasi 5 dengan nilai *fitness* MBO, kemudian pada populasi 21 dan 41 nilai *fitness*-nya memiliki selisih 0,2% dengan nilai *fitness* MBO. Pada kasus III, algoritme MBO juga memiliki nilai *fitness* terbaik yang sama pada setiap populasi dan nilai *fitness* PSO memiliki selisih 5,1% pada populasi 5, selisih 2,2% pada populasi 21, dan populasi 41 memiliki nilai *fitness* yang sama besar dengan nilai *fitness* MBO.

Tabel 13. Perbandingan total keuntungan (*fitness*) terbaik

Kasus	Populasi	*Fitness* terbaik (dalam rupiah)	
		MBO	PSO
I	5	888000	888000
	21	888000	888000
	41	888000	888000
II	5	4700000	4519000
	21	4700000	4696000
	41	4700000	4696000
III	5	6666500	6322500
	21	6666500	6581500
	41	6666500	6666500

SIMPULAN

Berdasarkan hasil penelitian ini dapat dilihat bahwa pada tiga kasus yang ada, algoritme MBO mempunyai nilai *fitness* yang mengarah ke nilai yang sama di setiap populasi. Sedangkan pada algoritme PSO, hanya kasus I yang mempunyai nilai *fitness* sama di setiap populasi. Pada kasus II, nilai *fitness* MBO lebih besar dibandingkan nilai *fitness* PSO dengan populasi 5 memiliki selisih 4%, populasi 21 dan 41 memiliki selisih 0,2%. Pada kasus III populasi 5 dan 21, nilai *fitness* MBO lebih besar dibandingkan nilai *fitness* PSO dengan selisih 5,1% pada populasi 5, selisih 2,2% pada populasi 21 dan populasi 41 memiliki nilai *fitness* yang sama besar. Algoritme MBO dan PSO dapat digunakan untuk mencari solusi *heuristic* dari masalah *knapsack* 0-1. Hasil penelitian tersebut menunjukkan algoritme *Migrating Birds Optimization* mempunyai nilai hasil fungsi objektif yang lebih baik dibandingkan dengan algoritme *Particle Swarm Optimization.*

DAFTAR PUSTAKA

Bhattacharjee, K. K., & Sarmah, S. P. (2014). Shuffled frog leaping algorithm and its application to 0/1 knapsack problem. *Applied Soft Computing, 19*, 252–263. https://doi.org/10.1016/j.asoc.2014.02.010

Bisilisin, F. Y., Herdiyeni, Y., & Silalahi, B. P. (2017). Optimasi K-Means Clustering Menggunakan Particle Swarm Optimization pada Sistem Identifikasi Tumbuhan Obat Berbasis Citra. *Jurnal Ilmu Komputer Dan Agri-Informatika*. https://doi.org/10.29244/jika.3.1.37-46

Chih, M., Lin, C.-J., Chern, M.-S., & Ou, T.-Y. (2014). Particle swarm optimization with time-varying acceleration coefficients for the multidimensional knapsack problem. *Applied Mathematical Modelling, 38*(4), 1338–1350. https://doi.org/10.1016/j.apm.2013.08.009

Duman, E., Uysal, M., & Alkaya, A. F. (2012). Migrating birds optimization: a new metaheuristic approach and its performance on quadratic assignment problem. *Information Sciences, 217*, 65-77. https://doi.org/10.1016/j.ins.2012.06.032

Iryanto, M. P., & Mardiyati, S. (2017). Penyelesaian { 0 , 1 } - knapsack problem dengan algoritma soccer league competition. *PRISMA, Prosiding Seminar Nasional Matematika*, 688–700. https://journal.unnes.ac.id/sju/index.php/prisma/article/view/21531

Kellerer, H., Pferschy, U., & Pisinger, D. (2004). Knapsack Problems. In Knapsack Problems. https://doi.org/10.1007/978-3-540-24777-7

Lalang, D., Silalahi, B. P., & Bukhari, F. (2018). Vehicle Routing Problem Time Windows Dengan Pengemudi Sesekali. *Journal of Mathematics and Its Applications, 17*(2), 87-99. https://doi.org/10.29244/jmap.17.2.87-99

Liang, Y., Liu, L., Wang, D., & Wu, R. (2010). Optimizing particle swarm optimization to solve knapsack problem. *Communications in Computer and Information Science, 105.* https://doi.org/10.1007/978-3-642-16336-4_58

Making, S. R. M., Silalahi, B. P., & Bukhari, F. (2018). Multi depot vehicle routing problem dengan pengemudi sesekali. *Journal of Mathematics and Its Applications, 17*(1), 75-86. https://doi.org/10.29244/jmap.17.1.75-86

Mayyani, H., Silalahi, B. P., & Aman, A. (2017). Frequency determination of bus rapid transit (BRT) applied on service system of trans Mataram metro bus to minimize the operational cost. *International Journal of Engineering and Management Research, 7*(6), 134–140. https://www.ijemr.net/DOC/FrequencyDeterminationOfBusRapidTransitBRTAppliedOnServiceSystemOfTransMataramMetroBusToMinimizeTheOperationalCost.pdf

Silalahi, B. P. (2019). Evaluation of interior-point method in Scilab. *IOP Conference Series: Earth and Environmental Science, 012040*. https://iopscience.iop.org/article/10.1088/1755-1315/299/1/012040/pdf

Silalahi, B. P., Bukhari, F., Aman, A., Khatizah, E., & Fahlevi, N. A. (2019). Comparison of interior point method execution time in solving linear optimization problems using Mathematica and Scilab. *International Journal of Statistics & Economics, 20*(4), 74–81. http://www.ceser.in/ceserp/index.php/bse/article/view/6196

Silalahi, B. P., Fathiah, N., & Supriyo, P. T. (2019). Use of ant colony optimization algorithm for determining traveling salesman problem routes. *Jurnal Matematika MANTIK, 5*(2), 100–111. https://doi.org/10.15642/mantik.2019.5.2.100-111

Silalahi, B. P., Fatihin, K., Supriyo, P. T., & Guritman, S. (2020). Algoritme sweep dan particle swarm optimization dalam optimisasi rute kendaraan dengan kapasitas. *Jurnal Matematika Integratif*, 16(1), 29-40. https://doi.org/10.24198/jmi.v16.n1.27474.29-40

Silalahi, B. P., Pertiwi, S. E., Mayyani, H., & Aliatiningtyas, N. (2020). Aplikasi zero-one goal programming dalam masalah pemilihan proyek pemasaran. *BAREKENG: Jurnal Ilmu Matematika Dan Terapan, 14*(3), 435–446. https://doi.org/10.30598/barekengvol14iss3pp435-446

Singh, R. P. (2011). Solving 0-1 Knapsack problem using Genetic Algorithms. *2011 IEEE 3rd International Conference on Communication Software and Networks, ICCSN 2011*, 591-595. https://doi.org/10.1109/ICCSN.2011.6013975

Ulker, E., & Tongur, V. (2017). Migrating birds optimization (MBO) algorithm to solve knapsack problem. *Procedia Computer Science, 111*, 71-76. https://doi.org/10.1016/j.procs.2017.06.012

Wihartiko, F. D., Buono, A., & Silalahi, B. P. (2017). Integer programming model for optimizing bus timetable using genetic algorithm. *IOP Conference Series: Materials Science and Engineering*. https://doi.org/10.1088/1757-899X/166/1/012016

Zhang, M., Tan, Y., Zhu, J., Chen, Y., & Chen, Z. (2020). A competitive and cooperative migrating birds optimization algorithm for vary-sized batch splitting scheduling problem of flexible Job-Shop with setup time. *Simulation Modelling Practice and Theory*, *100*, 102065. https://doi.org/10.1016/j.simpat.2019.102065

Zou, D., Gao, L., Li, S., & Wu, J. (2011). Solving 0-1 knapsack problem by a novel global harmony search algorithm. *Applied Soft Computing Journal*, *11*(2), 1556-1564. https://doi.org/10.1016/j.asoc.2010.07.019

Zouache, D., Moussaoui, A., & Ben Abdelaziz, F. (2018). A cooperative swarm intelligence algorithm for multi-objective discrete optimization with application to the knapsack problem. *European Journal of Operational Research, 264*(1), 74-88. https://doi.org/10.1016/j.ejor.2017.06.058