BAB IV
HASIL DAN PEMBAHASAN

1.1 Gambaran Umum Sistem

Pada subbab ini dibahas mengenai gambaran umum sistem pencarian rute optimum taksi menggunakan Algoritma *Particle Swarm Optimization*. Pada sistem ini diperlukan inputan node atau persimpangan awal dan akhir oleh pengguna. Sistem akan menghitung rute terbaik dari persimpangan awal ke persimpangan akhir. Data yang digunakan adalah data dari penelitian sebelumnya oleh Aditya Setiawan (2016) yang akan direpresentasikan ke sebuah *Graph* berbobot tidak berarah dengan bobot waktu tempuh kendaraan taksi. Terdapat dua mode tampilan untuk menampilkan hasil rute. Yang pertama adalah rute di *Google Maps* dan *Infografik* berupa urutan node-node yang membentuk jalur. Pengguna dari sisi *client* yaitu pengendara taksi. Sedangkan sisi server digunakan oleh *Admin* yang telah memiliki akses masuk ke dalam sistem.

1.2 Lingkungan Implementasi

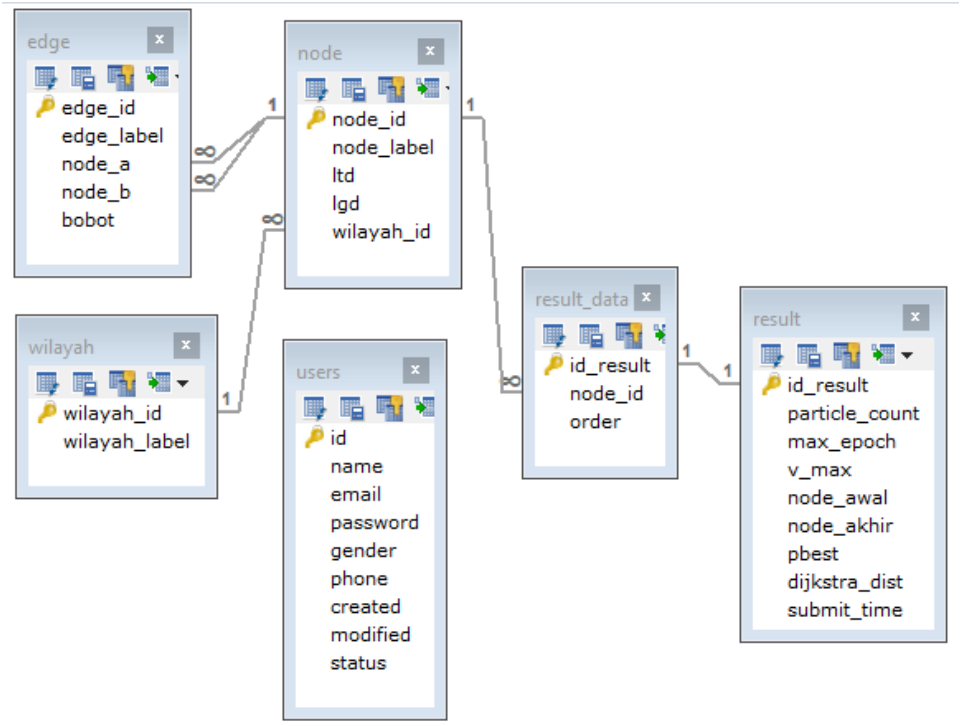
Sistem dirancang pada lingkungan berbasis sistem operasi Microsoft Windows 10 Pro Versi 64 bit. Sistem dirancang pada sebuah laptop yang memiliki kemampuan CPU Intel Core™ i3-4030U CPU @ 1.90Ghz, RAM 2 GB, dan kapasitas penyimpanan sebesar 700GB.

Pada tahap implementasi sistem, rancangan sistem akan diterjemahkan ke dalam kode program. Sistem ini akan dibangun menggunakan teknologi web pada sisi server dan teknologi *hybrid mobile* pada sisi client. Pada sisi server bahasa pemrograman yang akan digunakan adalah PHP 5.5.24, menggunakan framework CodeIgniter 3.1.4, dan basis data *MySql*. Pada sisi client, bahasa pemrograman yang akan digunakan adalah *JavaScript* menggunakan Framework *Ionic* versi 1.3. Web kemudian akan dilakukan hosting pada suatu *server* yang selanjutnya dapat diakses menggunakan jaringan internet.

1.3 Implementasi Database

Graph jaringan jalan di wilayah Denpsar Utara yang akan digunakan sebagai data yang selanjutnya diproses untuk mencari jalur optimum kendaraan umum taksi menggunakan Algoritma PSO akan diimplementasikan ke dalam bentuk database

relational pada MySQL. Gambar 4.1 merupakan skema basis data yang diturunkan dari ERD yang sebelumnya telah dirancang. Dari gambar dibawah ini terdapat 4 tabel basis data relational yang dihasilkan.



Gambar 4. 1 Implementasi Database

Adapun deskripsi tabel yang dihasilkan dari rangcangan ERD seperti pada gambar diatas, adaaah sebagai berikut :

1. Tabel *node*

Table node merupakan tabel yang dihasilkan dari entitas node yang memiliki atribut: node_id(*primary_key*) yang digunakan sebagai id yang bersifat unik, node_label yang menyimpan nama dari persimpangan, ltd yang menyimpan data latitude atau garis lintang persimpangan pada kordinat peta, lgd yang menyimpan data longitute atau garis bujur persimpangan pada kordinat peta yang berada di kawasan Denpasar Utara. Berikut adalah implementasi tabel node pada Gambar 4.2.

	#	Name	Type	Collation	Attributes	Null	Default	Extra
☐	1	node_id	int(11)			No	None	AUTO_INCREMENT
☐	2	node_label	varchar(255)	latin1_swedish_ci		No	None	
☐	3	ltd	double			No	None	
☐	4	lgd	double			No	None	
☐	5	wilayah_id	int(11)			No	None	

Gambar 4. 2 Implementasi Tabel Node

2. Tabel *edge*

Tabel edge merupakan tabel yang dihasilkan dari *entitas* edge yang memiliki atribut: *edge_id (primary_key)* yang digunakan sebagai id yang bersifat unik, *edge_label* yang menyimpan nama dari jalan yang menghubungkan antar persimpangan, *node_a* dan *node_b* merupakan persimpangan yang dihubungkan oleh edge_label, *bobot* merupakan waktu tempuh kendaraan taksi dari node_a ke node_b. Berikut adalah implementasi tabel edge pada Gambar 4.3.

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	edge_id	int(11)			No	None	AUTO_INCREMENT
2	edge_label	varchar(255)	latin1_swedish_ci		No	None	
3	node_a	int(11)			No	None	
4	node_b	int(11)			No	None	
5	bobot	float			No	None	

Gambar 4. 3 Implementasi Tabel Edge

3. Tabel *wilayah*

Tabel wilayah merupakan tabel yang menyimpan data wilayah dari setiap persimpangan pada tabel node. Tabel node berelasi dengan tabel wilayah dengan relasi satu ke banyak yang berarti satu wilayah bisa mewakili banyak node. Berikut adalah implementasi tabel wilayah pada Gambar 4.4.

Name	Type	Collation	Attributes	Null	Default	Extra
wilayah_id	int(11)			No	None	AUTO_INCREMENT
wilayah_label	varchar(255)	latin1_swedish_ci		No	None	

Gambar 4. 4 Implementasi Tabel Wilayah

4. Tabel *admin*

Tabel admin digunkan untuk validasi login dari admin. Tabel admin memiliki atribut *admin_id* (*primary*) yang digunakan sebagai id yang bersifat unik, *username*

dan *password* yang digunkan untuk validasi ke sistem agar dapat login. Berikut adalah implementasi tabel wilayah pada Gambar 4.5.

Name	Type	Collation	Attributes	Null	Default	Extra
admin_id	int(11)			No	*None*	AUTO_INCREMENT
username	varchar(255)	latin1_swedish_ci		No	*None*	
password	varchar(255)	latin1_swedish_ci		No	*None*	

Gambar 4. 5 Implementasi Tabel Admin

5. Tabel *Result*

Tabel Result digunakan untuk menyimpan data hasil proses pencarian rute terpendek menggunakan Algoritma PSO. *Record_id (primary)* yang digunakan sebagai id unik, *v_max* untuk menyinpan kecepatan maksimum yang iinputkan admin, *max_epoch* untuk menyimpan jumlah iterasi maksimal, *particle_count* untuk menyimpan jumlah partikel, *epoch_number* untuk melihat pada iterasi keberapa nilai fitness mencapai target, *shortest_route* untuk menyimpan urutan persimpangan yang membentuk rute optimum terbaik secara global, dan *shortest_distance* merupakan nilai fitness terbaik yang didapatkan sistem. Berikut adalah implementasi tabel *record* hasil pada Gambar 4.6.

Name	Type	Collation	Attributes	Null	Default	Extra
id_result	int(11)			No	*None*	AUTO_INCREMENT
particle_count	int(11)			No	*None*	
max_epoch	int(11)			No	*None*	
v_max	int(11)			No	*None*	
node_awal	int(11)			No	*None*	
node_akhir	int(11)			No	*None*	
pbest	int(11)			No	*None*	
dijkstra_dist	float			No	*None*	
submit_time	timestamp		on update CURRENT_TIMESTAMP	No	CURRENT_TIMESTAMP	ON UPDATE CURRENT_TIMESTAMP

Gambar 4. 6 Implementasi Tabel Result

6. Tabel *Result Data*

Tabel Result Data digunakan untuk menyimpan data detail hasil dari pencarian rute menggunakan Algoritma PSO dan Dijkstra. Id_result merupakan foreign key yang mengacu pada tabel Result. Node_id merupakan foreign key yang mengacu pada tabel node dan order merupakan urutan dari node yang membentuk rute perjalanan.

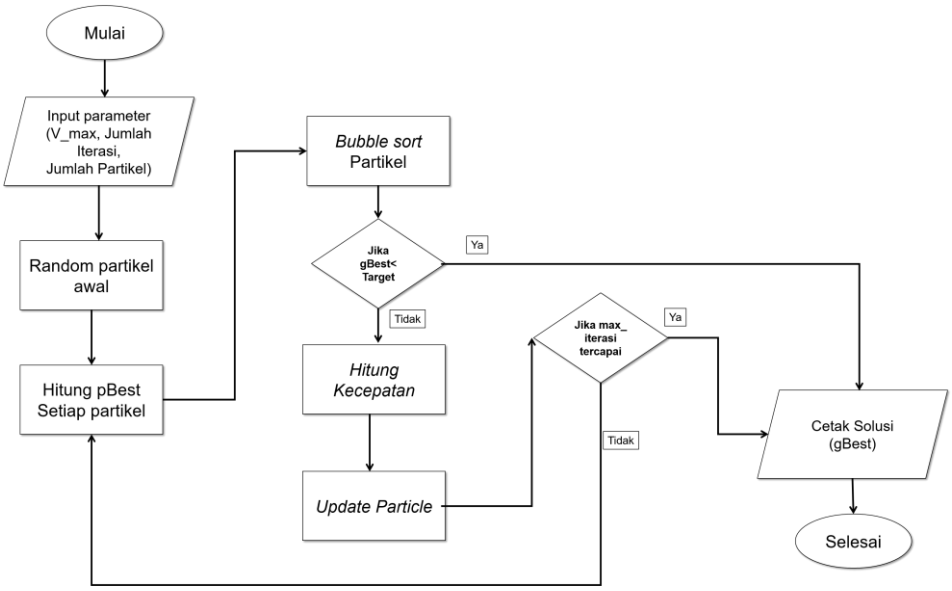
Name	Type	Collation	Attributes	Null	Default	Extra
id_result	int(11)			No	*None*	
node_id	int(11)			Yes	*NULL*	
order	int(11)			No	*None*	AUTO_INCREMENT

Gambar 4. 7 Implementasi Tabel Result Data

1.4 Implementasi Algoritma PSO

Masalah pencarian rute optimum kendaraan umum taksi di Kota Denpasar Utara akan direpresentasikan ke dalam sebuah graph tidak berarah dengan bobot berupa waktu tempuh per-menit dari kendaraan umum jenis taksi. Node merupakan persimpangan, dan jalan merupakah edge yang menghubungkannya. Solusi merupakan urutan persimpangan/node dari node awal ke node tujuan yang sebelumnya ditentukan oleh *user*. Nilai fitness merupakan total dari waktu tempuh dari node awal ke node tujuan. Untuk menyelesaikan permasalahan pencarian rute tersebut, Algoritma PSO akan menghasilkan partikel-partikel, yang merupakan kemungkinan solusi dari seluruh masalah pencarian rute yang selanjutnya nilai *global best* (gBest) akan mengindikasikan kemungkinan solusi terbaik global dari seluruh kemungkinan solusi. Algoritma PSO akan mencari dari seluruh solusi terbaik dari partikel (pBest) dan membandingkan satu dengan lainnya untuk mendapakan solusi terbaik global (gBest).

Algoritma Particle Swarm Optimization yang akan digunakan adalah dari penelitian (El-Sisi, 2014). Berikut merupakan flowchart dari Algoritma Particle Swarm Optimization yang digunakan pada penelitian ini:



Gambar 4. 7 Flowchart Algoritma PSO (El-Sisi, 2014)

Pada tahap inisialisasi, ditentukan terlebih dahulu parameter-parameter dari algoritma yang digunakan yaitu jumlah partikel, kecepatan maksimum, dan iterasi maksimum. Variable V_max digunkan untuk menentukan kecepatan maksimum yang diizinkan. Solusi direpresentasikan menjadi rute perjalanan sebagai sebuah array dengan *Id* dari persimpangan menunjukan urutan dari persimpangan-persimpangan yang dilalui. Nilai fitness merupakan jumlah seluruh bobot yang merupakan waktu tempuh total dari rute perjalanan.

Setiap partikel menyimpan variable *pBest*. Variabel *pBest* merepresentasikan nilai finess dari partikel. Setiap gBest dari patikel berganti ketika partikel menemukan nilai *pBest* yang dicapai lebih kecil.

Proses selanjutnya adalah proses update partikel dengan kecepatan dari setiap partikel dengan rumus:

$$V = \frac{V_{Max} * pBest}{WorstpBest} \quad \text{.....(El-Sisi, 2014)}$$

Nilai *worstpBest* merupakan nilai bobot terbesar dari keseluruhan partikel yang didapatkan dari hasil *BubbleSort*. Kecepatan mendefinisikan seberapa buruk solusi partikel. Semakin besar kecepatan, maka semakin sering partikel tersebut harus berpindah (El-Sisi, 2014). Kecepatan direpresentasikan dengan melakukan proses mengganti (*update*) node pada index ke-i dengan node yang baru. Proses ini

dilakukan untuk semua partikel kecuali yang terbaik secara global. Kriteria berhenti jika target telah dicapai atau maksimum iterasi sudah dicapai. Target didapatkan dari perhitungan menggunakan Algoritma Dijkstra. Berikut ini merupakah contoh perhitungan Manual dari Algoritma Particle Swarm Optimization:

1. Perhitungan Manual Agoritma Particle Swarm Optimization

a. Inisialisasi

Proses pertama yang dilakukan oleh Algoritma PSO adalah inisialisasi posisi awal dari node awal dan akhir dari inputan user. Misalkan node inputan user adalah 3 dan node akhir adalah 21, kecepatan maksimum adalah 8, jumlah partikel adalah 4, jumlah iterasi adalah 2.

Proses inisialisasi dimulai dengan proses random tetangga yang dimulai dari node 0 smpai node ke n-1. Akan dilalukan proses perulangan sebanyak jumlah total persimpangan yaitu 115, dan akan dicari tetangga-tetangga yang berhubungan dengan node ke-0, misalkan node 1 bertetatangga dengan 2, 3, dan 5 kemudian akan dilakukan proses *random* antara ketiga tetangga tersebut. Sebagai contoh hasil random adalah 3, maka node yang terbentuk adalah 1-3. Kemudian proses diulangi sampai terbentuk rangkaian node sebanyak n.

Contoh inisialisasi partikel:

Iteration number: 0
Particle **1** Route: 3 - 5 - 7 - 10 - 6 - 12 - 13 - 14 - 18 - 25 - 27 - 19 - 20 - 23 - 26 - 24 - 22 - 21
Particle **2** Route: 3 - 5 - 7 - 10 - 15 - 14 - 18 - 19 - 20 - 23 - 26 - 24 - 22 - 21
Particle **3** Route: 3 - 5 - 6 - 12 - 13 - 14 - 18 - 19 - 20 - 23 - 26 - 24 - 22 - 21
Particle **4** Route: 3 - 5 - 6 - 12 - 13 - 14 - 15 - 16 - 20 - 23 - 26 - 24 - 22 - 21

Iteration number: 1
Particle **1** Route: 3 - 5 - 7 - 10 - 15 - 14 - 18 - 19 - 20 - 23 - 26 - 24 - 22 - 21
Particle **2** Route: 3 - 5 - 6 - 12 - 13 - 14 - 18 - 19 - 20 - 23 - 26 - 24 - 22 - 21
Particle **3** Route: 3 - 5 - 6 - 12 - 13 - 14 - 15 - 16 - 20 - 23 - 26 - 24 - 22 - 21
Particle **4** Route: 3 - 5 - 7 - 10 - 6 - 12 - 13 - 14 - 18 - 25 - 27 - 19 - 20 - 23 - 26 - 24 - 22 - 21

b. Proses Utama Algoritma *Particle Swarm Optimization*

Kemudian masuk ke proses utama Algoritma PSO. Akan dilalukan perulangan sebanyak partikel untuk mencari nilai pBest masing-masing partikel. Nilai pBest didapatkan dengan cara menjumlahkan total waktu tempuh antar node, dari node node awal sampai ke node tujuan. Jika pBest partikel ke-i lebih kecil atau

sama dengan Target maka Global Best (gBest) atau solusi terbaik global (gBest) adalah pBest dari rute ke-i. Misalkan pBest tidak lebih kecil dari Target maka akan masuk ke proses berikutnya.

Contoh perhitungan nilai pBest:

Iteration number: 0
Particle **1** Route: 3- 5 - 7 - 10 - 6 - 12 - 13 - 14 - 18 - 25 - 27 - 19 - 20 - 23 - 26 - 24 - 22 - 21 **Distance:** 22.83
Particle **2** Route: 3 - 5 - 7 - 10 - 15 - 14 - 18 - 19 - 20 - 23 - 26 - 24 - 22 - 21 - **Distance:** 17.47
Particle **3** Route: 3 - 5 - 6 - 12 - 13 - 14 - 18 - 19 - 20 - 23 - 26 - 24 - 22 - 21 - **Distance:** 19.44
Particle **4** Route: 3 - 5 - 6 - 12 - 13 - 14 - 15 - 16 - 20 - 23 - 26 - 24 - 22 - 21 - **Distance:** 19.89

Iteration number: 1
Particle **1** Route: 3 - 5 - 7 - 10 - 15 - 14 - 18 - 19 - 20 - 23 - 26 - 24 - 22 - 21 - **Distance:** 17.47
Particle **2** Route: 3 - 5 - 6 - 12 - 13 - 14 - 18 - 19 - 20 - 23 - 26 - 24 - 22 - 21 - **Distance:** 19.44
Particle **3** Route: 3 - 5 - 6 - 12 - 13 - 14 - 15 - 16 - 20 - 23 - 26 - 24 - 22 - 21 - **Distance:** 19.89
Particle **4** Route: 3 - 5 - 7 - 10 - 6 - 12 - 13 - 14 - 18 - 25 - 27 - 19 - 20 - 23 - 26 - 24 - 22 - 21 - **Distance:** 22.83

Misalkan target yang dituju adalah 13.3 menit yang didapat dari hasil perhitungan Dijkstra jadi proses akan dilanjutkan karena target belum tercapai.

c. *Bubble Sort* Partikel

Proses selanjutnya adalah *Bubble Sort* dengan mengurutkan partikel dengan urutan node dengan pBest terbaik, atau bobot terkecil yang akan menjadi partikel ke-0 begitu pun seterusnya sampai dengan partikel ke-n. Partikel ke-0 akan menjadi gBest tiap iterasi.

Contoh proses *Bubble Sort*:

Iteration number: 0

Particle **2** Route: 3 - 5 - 7 - 10 - 15 - 14 - 18 - 19 - 20 - 23 - 26 - 24 - 22 - 21 - **Distance:** 17.47
Particle **3** Route: 3 - 5 - 6 - 12 - 13 - 14 - 18 - 19 - 20 - 23 - 26 - 24 - 22 - 21 - **Distance:** 19.44
Particle **4** Route: 3 - 5 - 6 - 12 - 13 - 14 - 15 - 16 - 20 - 23 - 26 - 24 - 22 - 21 - **Distance:** 19.89

Particle **1** Route: 3 - 5 - 7 - 10 - 6 - 12 - 13 - 14 - 18 - 25 - 27 - 19 - 20 - 23 - 26 - 24 - 22 - 21 **Distance:** 22.83

pBest: 17.47

Iteration number: 1
Particle **1** Route: 3 - 5 - 7 - 10 - 15 - 14 - 18 - 19 - 20 - 23 - 26 - 24 - 22 - 21 - **Distance:** 17.47
Particle **2** Route: 3 - 5 - 6 - 12 - 13 - 14 - 18 - 19 - 20 - 23 - 26 - 24 - 22 - 21 - **Distance:** 19.44
Particle **3** Route: 3 - 5 - 6 - 12 - 13 - 14 - 15 - 16 - 20 - 23 - 26 - 24 - 22 - 21 - **Distance:** 19.89

Particle **4** Route: 3 - 5 - 7 - 10 - 6 - 12 - 13 - 14 - 18 - 25 - 27 - 19 - 20 - 23 - 26 - 24 - 22 - 21 - **Distance:** 22.83

pBest: 17.47

d. Hitung Kecepatan Partikel

Selanjutnya akan dicari kecepatan dari masing-masing partikel. Kecepatan didapatkan dari rumus:

$$V = \frac{V_{Max} * pBest}{WorstpBest}$$

Kecepatan digambarkan dengan proses suatu partikel mendekati solusi terbaik dengan melalukan proses *update* node pada partikel. Kecepatan merupakan probabilitas seberapa banyak proses *update* dilakukan (**El-Sisi, 2014**). Semakin besar v, maka semakin banyak update dilakukan. Misalkan diketahui pada iterasi pertama V_max = 13.3, pBest = 17.47 karena partikel pertama pada iterasi ke-0 yang merupakah hasil dari *bubblesort*, WortstpBest = 22.83 yang merupakan partikel terakhir hasil dari *bubblesort,* jadi nilai v masing-masing partikel ditunjukan oleh contoh perhitungan berikut.

Contoh perhitungan kecepatan:

Iteration number: 0

Particle **2** Route: 3 - 5 - 7 - 10 - 15 - 14 - 18 - 19 - 20 - 23 - 26 - 24 - 22 - 21 - **Distance:** 17.47

V = 17.3806

Particle **3** Route: 3 - 5 - 6 - 12 - 13 - 14 - 18 - 19 - 20 - 23 - 26 - 24 - 22 - 21 - **Distance:** 19.44

V= 15.61929

Particle **4** Route: 3 - 5 - 6 - 12 - 13 - 14 - 15 - 16 - 20 - 23 - 26 - 24 - 22 - 21 - **Distance:** 19.89

V = 15.26591

Particle **1** Route: 3 - 5 - 7 - 10 - 6 - 12 - 13 - 14 - 18 - 25 - 27 - 19 - 20 - 23 - 26 - 24 - 22 - 21
Distance: 22.83

V = 13.3
pBest: 17.47

Iteration number: 1
Particle **1** Route: 3 - 5 - 7 - 10 - 15 - 14 - 18 - 19 - 20 - 23 - 26 - 24 - 22 - 21 - **Distance:** 17.47
Particle **2** Route: 3 - 5 - 6 - 12 - 13 - 14 - 18 - 19 - 20 - 23 - 26 - 24 - 22 - 21 - **Distance:** 19.44
Particle **3** Route: 3 - 5 - 6 - 12 - 13 - 14 - 15 - 16 - 20 - 23 - 26 - 24 - 22 - 21 - **Distance:** 19.89

Particle **4** Route: 3 - 5 - 7 - 10 - 6 - 12 - 13 - 14 - 18 - 25 - 27 - 19 - 20 - 23 - 26 - 24 - 22 - 21 -
Distance: 22.83

pBest: 17.47

Untuk proses update partikel dilakukan perulangan sebanyak jumlah partikel. Kecepatan menentukan seberapa banyak dilakukan proses update partikel. Misalkan kecepatan 5.6 yang akan dibulatkan menjadi 6. Jadi akan dilakukan 6 kali proses update node pada partikel ke-i sampai ke-n. Partikel ke-0 tidak mengalami update karena merupakan gBest (Global Best).

e. Proses Update Partikel.

Proses update dilakukan dengan memperbarui node pada partikel ke-i. Index dari node yang akan diperbarui didapatkan dari bilangan random. Untuk validasi dari urutan rute, akan dilakukan kembali proses pembentukan kembali rute pada partikel dengan membangkitkan bilangan random dari 1 sampai jumlah node. Hasil bilangan random r merupakan index dari node pada partikel yang akan diperbarui dengan cara mengganti tetangga dari node dengan index r. Misalkan hasil bilangan random adalah 5 maka node dengan indek ke-5 akan diperbarui.

Sebagai contoh index ke-5 pada partikel 1 adalah 14 yang merupakan node tetangga dari 15. Kemudian akan dicari tetangga selain 14 dari node 15, misalkan didapatkan tetangga selain 14 adalah 17,18,dan 19. Akan dilakukan random ketiga nilai tersebut. Misalkan nilai yang didapatkan adalah 17, maka akan dilalukan proses pembentukan jalur dengan mengganti node ke-5 dengan 17, akan dihasilkan rute sebagai beriku:

Particle **awal**:
3 - 5 - 7 - 10 - 15 - 14 - 18 - 19 - 20 - 23 - 26 - 24 - 22 - 21
Setelah proses pergantian:
3 - 5 – 7 – 10 – 17 – 19 – 21 – 25 – 22 - 21

Kemudian proses diulangi sampai iterasi maksimum tercapai atau nilai Global Best mencapai Target.

Adapun penggalan *source code* untuk masing-masing fungsi dari Algoritma PSO yang digunakan ditunjukan pada sub bab berikut:

1. *Sourcecode* Proses Inisialisasi Parameter

Sourcecode 4. 1 Source code Proses Inisialisasi Parameter

```
1  public function set_parameter()
2  {
     $init = $this->input->get();
```

```
3   if($init['particle_count'] > 0){
        $this->PARTICLE_COUNT = $init['particle_count'];
4   }else{
        $this->PARTICLE_COUNT = 10;
5   }
    if($init['v_max'] > 10 || $init['v_max'] < 1){
        $this->V_MAX = 8;
6   }else{
        $this->V_MAX = $init['v_max'];
7   }
    if($init['max_epoch'] > 100){
        $this->MAX_EPOCH = $init['max_epoch'];
8   }else{
        $this->MAX_EPOCH = 10;
9   }
10
    if (!isset($init['city1']) && !isset($init['city2'])) {
11      $this->cityA = 1;
        $this->cityB = 2;
12  }
    else
13  {
        $this->cityA = $init['city1'];
14      $this->cityB = $init['city2'];
    }
15  $raw=file_get_contents("http://arisudana.web.id/ci2/dijkst
    ra/get3?city1=".$this->cityA."&city2=".$this->cityB."");
16
    $dijkstra = json_decode($raw);
17
    if (!isset($dijkstra->route)) {
18  echo "<script>alert('Target Belum diset')</script>";
    }
19  else
    {
20      $this->dijkstra_route = $dijkstra->route;
    }
21  if (!isset($dijkstra->distance)) {
        $this->TARGET = 8.98;
22  }
    else
23  {
        $this->TARGET = $dijkstra->distance;
24  }
25
26
27
28
29
30
31
32
33
34
35
```

```
36
37
38
39
40
41
42
43
44
45
46
47
48
```

Implementasi source code 4.1 merupakan tahap implementasi dari tahap inisialisi yang di dalamnya terdapat pengambilan parameter-parameter yang bersumber dari inputan user. Parameter-parameter tersebut adalah jumlah partikel (*PARTICLE_COUNT*), kecepatan maksimal (*V_MAX*), maksimum iterasi (*MAX_EPOCH*), target solusi (*TARGET*), persimpangan awal (*cityA*), dan persimpangan tujuan (*cityB*). Target yang digunakan bersumber dari Algoritma Dijkstra yang diakses pada `file_get_contents` untuk mendapatkan hasil perhitungan dari Algoritma Dijkstra dengan mengirimkan parameter *cityA* dan *cityB.*

2. *Sourcecode* Proses Inisialisasi Partikel

Sourcecode 4. 2 Inisialisasi Partikel

```
1  for($i = 0; $i < $this->PARTICLE_COUNT; $i++){
2     $newParticle = new Particle(); // membuat object baru
3     $newParticle->setlabel($i+1); // label atau nama
4  particle
5  for($j = 0; $j < $this->CITY_COUNT; $j++){
6     $newParticle->setData($j, NULL);
7     $newParticle->setRute($j, NULL);
```

```
8   }
9      array_push($this->particles, $newParticle
10     $value = array();
11  while($this->validation($i, $this->cityB) != 1)
12  {
13      $value = $this->randomlyArrange($i, $this->cityA,
14  $this->cityB);
15  }
16  foreach ($value as $key => $nilai) {
17    $this->particles[$i]->setRute($key, $nilai);
18  }
    }
```

Implementasi *source code* 4.2 merupakan proses inisialiasi dari partikel yang merupakan calon solusi dari permasalahan. Akan dideklarasikan sebuah *object Partikel* yang memiliki *label*, kecepatan, bobot serta rute perjalanan. Kemudian akan dipanggil fungsi *randomlyArrange()* untuk mengacak posisi setiap node untuk merepresentasikan solusi dari masalah. Akan dilakukan perulangan, selama partikel ke-i tidak mengandung node tujuan, maka akan dilakukan proses *randomlyArrange()*, sampai pada partikel terdapat node tujuan.

3. *Sourcecode* Proses Random Partikel

Sourcecode 4. 3 Random Partikel

```
1   private function randomlyArrange($index, $start, $finish)
2   {
3   if(!isset($cityA))
4   {
5     $cityA = $start;
6   }
7   $cityB = $finish;
8   $done = FALSE;
9   $temp = array();
10  $flipped = array();
```

```
$history = array();
$history[] = $cityA;
$temp[0] = $start;
if (isset($this->graph2[$cityA])) {
    $startnode = $this->graph2[$cityA];
}
else if(isset($this->graph3[$cityA]))
{
  $startnode = $this->graph3[$cityA];
}
$this->particles[$index]->setData(0, $cityA); // awal
rute
while($done != 1){
if (!in_array($cityB, $temp)) {
if (isset($this->graph2[$cityA])) {
     $flipped = $this->graph2[$cityA];
}
else if (isset($this->graph3[$cityA])) {
      $flipped = $this->graph3[$cityA];
}
else
{
     $done = 1;
}
$cropped = array_diff($flipped, $history);
if (empty($cropped)) {
$done = 1;
}
else
{
$chek = array_rand($cropped);
if ($cropped[$chek] == $cityB) {
$temp[] = $cropped[$chek];
$cityA = $cropped[$chek];
```

```
44  $history[] = $cropped[$chek];
45  $done = 1;
46  }
47  else
48  {
49    $temp[] = $cropped[$chek];
50    $cityA = $cropped[$chek];
51    $history[] = $cropped[$chek];
52    $done = 0;
53  }
54  }
55  }
56  else
57  {
58  $done = 1;
59  }
60  }
61  for ($i=1; $i < count($temp); $i++) {
62    $this->particles[$index]->setData($i, $temp[$i]);
63  }
64  return $temp;
65  }
66
```

Source code 4.3 merupakan implementasi proses random partikel. Fungsi *randomlyArrange()* menerima parameter *$index*, *$start*, dan *$finish*. Variabel *$start* merupakan node awal dari rute yang akan terbentuk, *$finish* merupakah node akhir dari rute yang akan terbentuk dan *$index* merupakan index dari partikel. Proses yang terjadi pada fungsi ini adalah melakukan proses pencarian rute yang tersedia pada graph, dari node awal ke node tujuan, dengan merandom node-node yang berada di antara node awal dan tujuan tersebut. Pada proses ini menggunakan beberapa variable yaitu *$temp*, untuk menyimpan node sementara yang akan menjadi urutan node ke tujuan. Variabel *$temp* merupakan array dengan index ke-

0 diisi nilai node awal, karena merupakan awal dari rangkaian node. Kemudian akan masuk ke dalam perulangan, selama nilai *$done* tidak sama dengan TRUE, maka perulangan akan dilakukan. Terdapat beberapa proses logika IF pada perulangan. Pertama akan dicek apakah array *$temp* sudah menyimpan node tujuan, jika belum maka akan dilakukan pengecekan kedua yaitu apakah terdapat tetangga dari *$cityA*, jika ada maka tetangga dari *$cityA* akan disimpan ke dalam array *$flipped*. Jika tidak, maka perulangan akan berakhir, karena *$cityA* tidak memiliki tetangga. Kemudian akan akan dilakukan fungsi PHP *array_diff* yang akan menghasilkan irisan dari history dari rute yang telah ditemukan dengan tetangga-tetangga dari cityA, ini bertujuan agar tidak ada node yang sama pada partikel. Hasilnya akan disimpan didalam array *$cropped*. Kemudian nilai atau value dari array $cropped akan dirandom untuk mencari node selanjutnya yang disimpan didalam variable *$chek*. Kemudian akan ada pengecekan kembali, jika array $cropped tidak kosong, maka node selanjutnya akan diisi dengan variable *$chek.* Begitupun seterusnya sampai nilai $done == TRUE.

4. *Sourcecode* Proses Utama PSO

Sourcecode 4. 4 Proses Utama PSO

```
1  $aParticle = null;
2  $epoch = 0;
3  $done = FALSE;
4  $this->initialize();
5  while(!$done)
6  {
7  if($epoch < $this->MAX_EPOCH){
8  echo "<br><br>Iteration number: ".$epoch."<br>";
9  for($i = 0; $i < $this->PARTICLE_COUNT; $i++){
10 $aParticle = $this->particles[$i];
11 echo "Particle <strong>".$aParticle->label()."</strong>
12 ";
13 echo "Route: ";
14 for($j = 0; $j < $this->CITY_COUNT; $j++){
15 if ($aParticle->rute($j) != NULL) {
```

```
16  echo $aParticle->rute($j)." - ";
17  }
18  }
19  $this->getTotalDistance($i);
20  echo "<strong>Distance: </strong>".$aParticle-
21  >pBest().'<br>';
22  if($aParticle->pBest() <= $this->TARGET){
    $this->shortestDistance = $aParticle->pBest();
23  for($j = 0; $j < $this->CITY_COUNT; $j++) {
24   $this->shortestRoute.= $aParticle->rute($j) . ",";
25  }
26  $done = TRUE;
27  }
28  }
29  $this->bubbleSort();
30  $this->getVelocity();
31  $this->updateParticle();
32  $epoch++;
33  if ($this->particles[0]->pBest() == $this->TARGET) { //
34  jika gBest sudah mencapai target, hentikan iterasi
35  $done = TRUE;
36  }
37  }else{
38  $done = TRUE;
39  }
40  }
41
```

Implementasi dari proses utama Algoritma PSO pada source code 4.4 ini dimulai dengan proses mencetak node-node yang dihasilkan dari proses inisialisasi yang kemudian akan didapatkan total nilai fitnessnya oleh fungsi *getTotalDistance*(). Selanjutnya akan dilakukan pengecekan, jika nilai dari fitness partikel ke-i yang merupakan pBest atau solusi terbaik partikel ke-i lebih kecil daripada gBest, maka nilai dari gBest merupakan pBest partikel ke-i. Jika tidak

maka nilai gBest tidak berubah. Kemudian akan dilakukan proses *bubbleSort()* untuk mengurutkan partikel berdasarkan nilai fitness terkecil ke terbesar. Selanjutnya akan dipanggil proses *getVelocity()* untuk mendapatkan kecepatan dari masing-masing partikel, dan *updateParticle()* untuk memberbarui rute dari partikel dengan menggunakan kecepatan. Jika iterasi mencapai iterasi maksimum atau nilai pBest lebih kecil daripada gBest maka perulangan akan dihentikan.

5. *Sourcecode* Proses Menghitung Kecepatan

Sourcecode 4. 5 Sourcecode Proses Menghitung Kecepatan

```
1  private function getVelocity(){
2    $worstResult = $this->particles[$this->PARTICLE_COUNT -
3  1]->pBest();
4    for($i = 0; $i < $this->PARTICLE_COUNT; $i++){
5  $vValue = ($this->V_MAX * $this->particles[$i]->pBest())
   / $worstResult;
6  if($vValue > $this->V_MAX){
7     $this->particles[$i]->setVelocity($this->V_MAX);
8  }elseif($vValue < 0.0){
9     $this->particles[$i]->setVelocity(0.0);
10 }else{
11    $this->particles[$i]->setVelocity($vValue);
12 }}}
```

Implementasi dari proses mencari kecepatan pada *Sourcecode* 4.5 dimulai dengan menghitung nilai fitness terburuk dari partikel yaitu partikel terakhir (setelah diurutkan dari fitness terkecil ke terbesar) dan disimpan di variable *$worstResult*. Selanjutnya masuk ke proses perulangan sebanyak jumlah partikel. Akan dihitung kecepatan dengan cara mengalikan fitness partikel saat ini dengan kecepatan maksimal, kemudian dibagi dengan fitness terburuk *($worstResult).* Hasil dari perhitungan tersebut akan disimpen ke variable *$vValue*. Kemudian akan dicek menggunakan IF, jika *$vValue* lebih besar daripada kecepatan maksimum, maka nilai kecepatan menjadi kecepatan maksimum agar hasil kecepatan tidak lebih besar daripada kecepatan maksimum. Jika $vValue lebih kecil daripada 0,

maka kecepatan menjadi 0 agar kecepatan tidak lebih dari 0. Jika $vValue tidak lebih kecil dari 0 dan tidak lebih besar dari kecepatan maksimum, maka kecepatan partikel ke-i akan diset dengan nilai *$vValue.*

6. *Sourcecode* Proses Update Partikel

Sourcecode 4. 6 Sourcecode Proses Update Partikel

```
1  private function updateParticle(){
2  $listSize = count($this->particles);
3  for($i = 1; $i < $listSize - 1; $i++){
4  $changes = (int)(floor(abs($this->particles[$i]-
5  >velocity())));
6  $city_count = count($this->particles[$i]->mData);
7  for($j = 0; $j < $changes; $j++){
8  $this->updateRoute(mt_rand(1, $city_count),$i);
9  }
10 $this->getTotalDistance($i);
11 }
   }
```

Implementasi dari proses mencari kecepatan pada Source Code 4.6 dimulai dengan menghitung nilai dari variable *$change* yaitu *variabel* yang menunjukan seberapa banyak suatu partikel harus diubah. Variabel *$change* merupakan nilai absolut dan pembulatan dari kecepatan agar nilai kecepatan menjadi bilangan bulat darn tidak bernilai minus. Fungsi *updateRoute()* akan dieksekusi sebanyak nilai *$change.* Dan kemudian akan dihitung kembali total nilai bobot dari partikel.

***Sourcecode* Proses Menghitung Total Nilai Fitness**

Sourcecode 4. 7 Sourcecode Proses Hitung Total Fitness

```
1  private function getTotalDistance($index){
2     $thisParticle = $this->particles[$index];
3     $thisParticle->setpBest(0.0);
4     for($i = 0; $i < $this->CITY_COUNT; $i++){
5       if(($this->CITY_COUNT - 1) == $i){
6
```

```
7       $thisParticle->setpBest($thisParticle->pBest() +
8   $this->getDistance($thisParticle->data($this->CITY_COUNT
    - 1), $thisParticle->data(0)));
9        }else{
10      $thisParticle->setpBest($thisParticle->pBest() +
11  $this->getDistance($thisParticle->data($i),
12  $thisParticle->data($i+1)));
13  }
    }
    }
```

Implementasi dari proses menghitung total fitness pada Source code 4.7 dimulai dengan mengecek setiap index node pada partikel, jika node bukan node terakhir pada urutan node, maka maka nilai fitness partikel ke-i akan ditambah dengan partikel partikel ke-i +1 atau partikel selanjutnya. Jika node dengan index terakhir, maka node tersebut akan ditambah dengan fitness awal.

7. *Sourcecode* Proses Menghitung Nilai Fitness

Sourcecode 4. 8 Hitung Nilai Fitness

```
1   private function getDistance($firstCity, $secondCity){
2   if (isset($this->bobot[$firstCity][$secondCity])) {
3   $cityA = $this->bobot[$firstCity][$secondCity];
4   }
5   else{
6    $cityA = 1000;
7   }
8   return $cityA;
9   }
```

Implementasi dari proses mendapatkan nilai fitness berupa waktu tempuh dari dua titik ditunjukan pada Sourcecode 4.8. Pertama dilakukan pengecekan apakah nilai dari parameter *$firstCity* dan *$seconCity* ada, jika ada maka ambil data bobot dari function $this-bobot antara dua parameter tersebut. Jika bobot tidak ada maka nilai bobot sama dengan 1000 untuk menandai jika rute yang terbentuk tidak valid.

8. *Sourcecode* Objek Partikel

Sourcecode 4. 9 Objek Partikel

```
1   class Particle
2   {
3       function __construct()
4       {
5           $this->mData = [];
6           $this->mpBest = 0;
7           $this->mVelocity = 0.0;
8           $this->label = 0;
9       }
10      public function compareTo($that){
11          if($this->pBest() < $that->pBest())
12              return -1;
13          elseif($this->pBest() > $that->pBest())
14              return 1;
15          else
16
17              return 0;
18      }
19
20      public function data($index){
21
22          return $this->mData[$index];
23      }
24
24      public function setData($index, $value){
25
26          $this->mData[$index] = $value;
26      }
27
28      public function pBest(){
29
30          return $this->mpBest;
31      }
32
33      public function setpBest($value){
34
35          $this->mpBest = $value;
36      }
37
38      public function velocity(){
39
```

```
40          return $this->mVelocity;
        }
        public function setVelocity($velocityScore){
            $this->mVelocity = $velocityScore;
        }
        public function label(){
            return $this->label;
        }
        public function setlabel($value){
            $this->label = $value;
        }
}
```

Implementasi objek sebuah partikel pada source code 4.9 memiliki beberapa atribut dan method. Atribut *mData*() merupakan urutan persimpangan yang membentuk calon solusi jalur optimal. Variabel *mpBest* menyimpan nilai fitness terbaik dari partikel (*pBest*). Variabel *mVelocity* menyimpan kecepatan partikel dan *label()* untuk memberikan index atau nama kepada partikel. Setiap partikel juga memiliki method yang dapat digunakan yaitu *compareTo()* untuk membandingkan antara nilai fitness satu partikel dengan yang lain, *data()* untuk mendapatkan node pada partikel. Method *pBest()* untuk mendapatkan nilai pBest dari partikel. Method *setpBest()* untuk mendapatkan memasukan nilai pBest pada partikel. Method *Velocity()* untuk mendapatkan nilai kecepatan partikel. Method *setVelocity()* untuk memasukan nilai kecepatan untuk partikel. Method *Label()* untuk mendapatkan label dari partikel dan Method setlabel() untuk set label kepada partikel.

9. Proses Mendapatkan Bobot dari Graph

Sourcecode 4. 10 Mendapatkan Bobot dari Graph

```
1   public function bobot()
2   {
3       $graph = array();
4       $this->load->model('Edge');
5       $edge = new Edge();
```

```
6       $data = json_decode($edge->bobot(), true);
7       foreach ($data as $id => $row) {
8          $graph[$row['node_id']][$row['node_b']] =
9   $row['bobot'];
10      }
11        return $graph;
12      }
13      public function tetangga()
14      {
15        $graph = array();
16        $this->load->model('Edge');
17
18        $edge = new Edge();
19
20        $data = json_decode($edge->bobot(), true);
21      foreach ($data as $id => $row) {
22
23        $graph[$row['node_id']][$row['node_b']] =
24  $row['node_b'];
        }
            return $graph
    }
```

Implementasi source code 4.10 merupakan function untuk mendapatkan data yang berasal dari query database untuk digunakan sebagai representasi graph dari wilayah Denpasar Utara yang selanjutnya dapat diproses oleh Algoritma PSO. Function Bobot() untuk mendapatkan bobot dari graph, dan Graph() untuk mendapatkan tetangga dari setiap node pada graph.

10. Proses Cetak Solusi Terbaik

Sourcecode 4. 11 Cetak Solusi Terbaik

Baris	Source code
1	`public function printBestSolution(){`
2	`if($this->particles[0]->pBest() <= $this->TARGET){`
3	`echo "<h4>Target Reached</h4>";`
4	

```
5           }else{
6               echo "<h4>Target not Reached</h4>";
7           }
8           for($i = 0; $i < $this->CITY_COUNT; $i++){
9               echo $this->particles[0]->data($i)."-";
10          }
        }
```

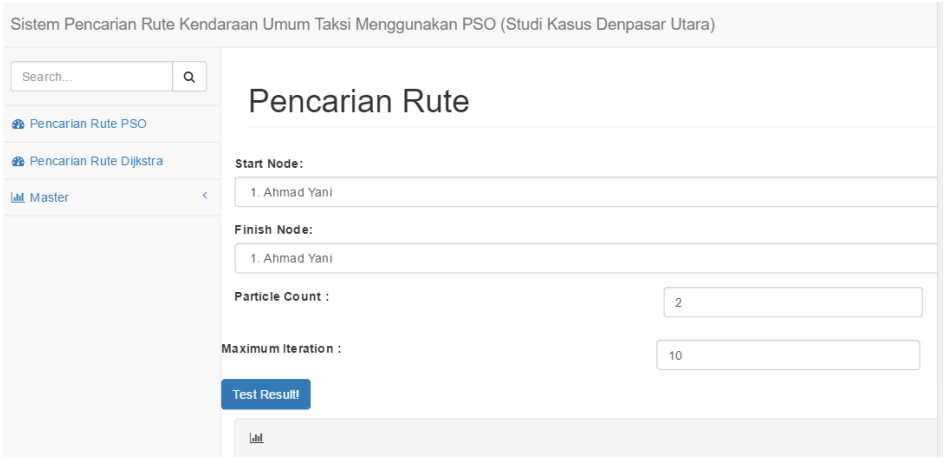
Implementasi source code 4.11 merupakan proses terakhir untuk mencetak solusi terbaik (gBest) yang merupakan partikel dengan index 0, karena sudah mengalami proses Bubblesort dari fitness terkecil ke terbesar, sehingga gBest merupakan partikel dengan index terkecil yaitu 0.

1.5 Impementasi Antarmuka

Pada subbab ini akan dijelaskan implementasi antarmuka sistem yang sudah dirancang. Antarmuka sistem diimplementasikan ke dalam HTML dan CSS dengan menggunakan Bootstrap 3 pada Adminitrator, sedangkan user menggunakan *Framework Ionic*. Implementasi antarmuka dibagi menjadi dua yaitu implementasi untuk admin dan untuk user. Berikut akan dijelaskan hasil dari implementasi rancangan yang sebelumnya sudah dibuat.

1.5.1 Impelementasi Antarmuka Pada Admin

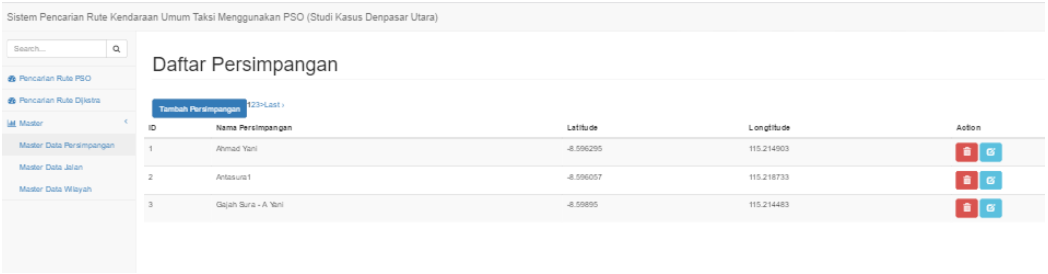
a. Implementasi Halaman Awal (Home)



Gambar 4. 8 Halaman Awal (Home)

Tampilan Gambar 4.8 merupakan tampilan halaman awal (home) dari sistem administrator ketika admin berhasil login ke dalam sistem. Terdapat form yang dapat diinputkan admin jika akan melakukan proses pencarian rute berupa node awal, node akhir, jumlah partikel, jumlah iterasi maksimum. Yang kemudian hasil dari pencarian rute akan ditampilkan pada google maps .

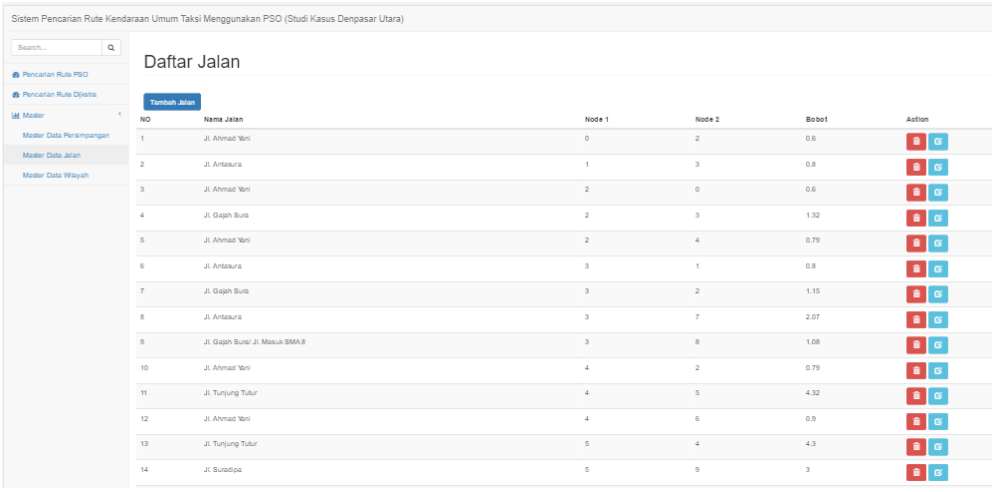
a. Implementasi View Data Persimpangan



Gambar 4. 9 Halaman View Data Persimpangan

Tampilan Gambar 4.9 merupakan tampilan untuk melihat seluruh data persimpangan. Admin juga bisa menambah data baru, edit data yang sudah ada serta delete data.

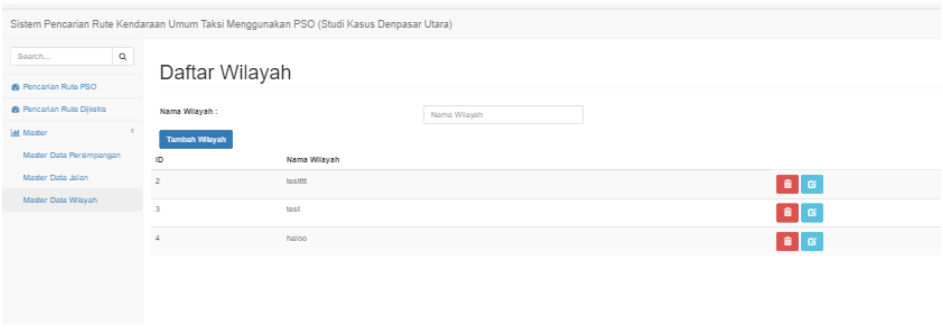
b. Implementasi View Data Jalan



Gambar 4. 10 Halaman View Data Jalan

Tampilan 4.10 merupakan tampilan untuk melihat seluruh data jalan. Admin juga bisa menambah data baru, edit data yang sudah ada serta delete data.

c. Implementasi View Data Wilayah



Gambar 4. 11 Halaman View Data Wilayah

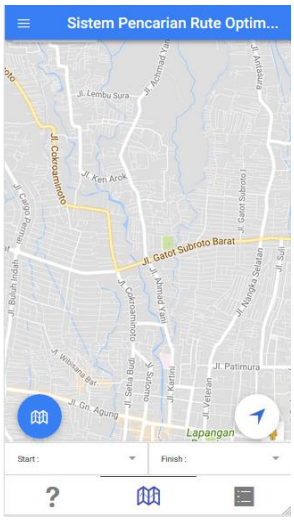
Tampilan Gambar 4.11 merupakan tampilan untuk melihat seluruh data wilayah. Admin juga bisa menambah data baru, edit data yang sudah ada serta delete data.

1.5.2 Implementasi Antarmuka pada User

Terdapat 3 tampilan utama/form dari aplikasi untuk User ini yaitu, *form tampilan rute, form seluruh node,dan form home.*

a. Form Tampilan Rute

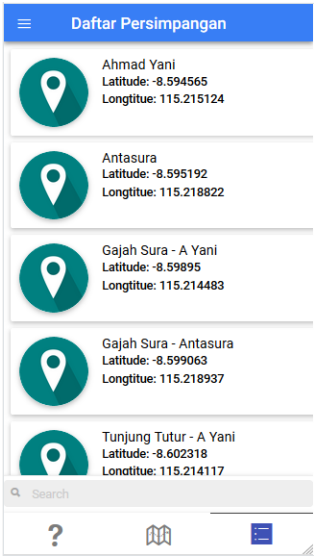
Pada Gambar 4.12 form tampilan rute user dapat memilih titik awal dan akhir untuk selanjutnya diproses oleh Algoritma Dijkstra untuk mendapatkan kemungkian jalur terbaik. User memilih titik awal dan akhir dengan memilih *list tab*. Kemudian setelah user memilih tombol *search*, rute akan ditampilakan berupa titik-titik yang dihubungkan menggunakan garis yang merepresentasikan jalur optimum dari kendaraan taksi.



Gambar 4. 12 Form Tampilan Rute

b. Form Seluruh Node

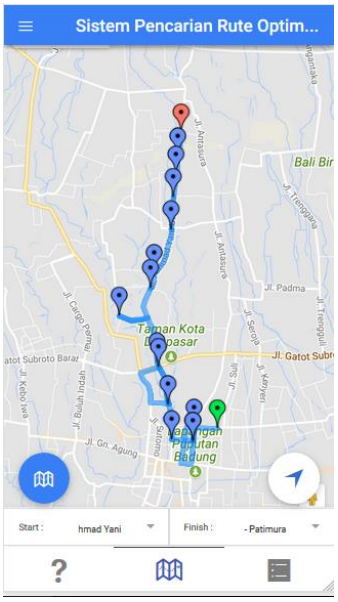
Pada form ini akan ditampilan seluruh node yang merupakan persimpangan di wilayah Denpasar Utara. Pada bagian bawah terdapat *form search* yang dapat digunakan untuk mencari node yang diinginkan.



Gambar 4. 13 Form Tampilan Seluruh Node

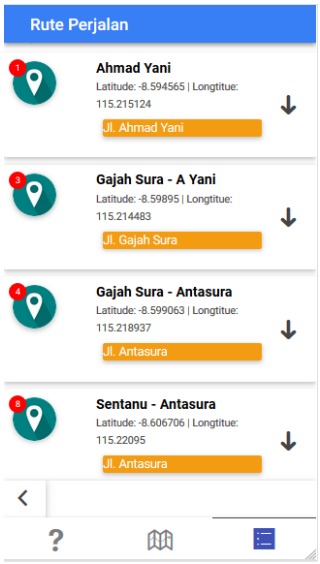
c. Form Hasil Pencarian Rute

Gambar 4.14 merupakan hasil dari pencarian rute yang ditampilkan dengan titik-titik yang dihubungkan dengan garis yang merupakan fitur dari API Google Maps.



Gambar 4. 14 Form Tampilan Hasil Pencarian Rute Tampilan Google Maps

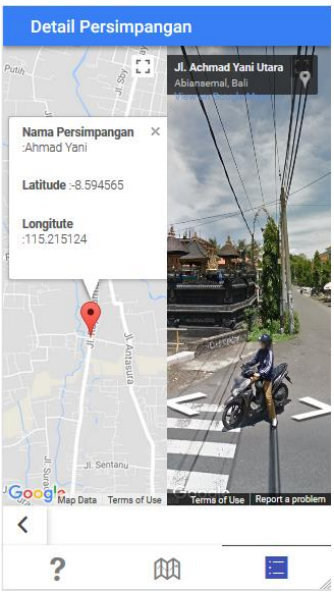
Gambar 4.15 merupakan hasil dari pencarian rute yang ditampilkan berupa urutan rute perjalanan yang ditampilakan secara berurutan dari atas ke bawah.



Gambar 4. 15 Form Tampilan Hasil Pencarian Rute Tampilan Infografik

d. Form Detail Persimpangan

Gambar 4.16 merupakan detail dari persimpangan yang muncul jika user memilih salah satu dari persimpangan dari list persimpangan atau hasil pencarian rute mode Infografik. Akan ditampilkan lokasi dari persimpangan di *Google Maps* serta tampilan dari *Google Street View* dari persimpangan tersebut.



Gambar 4. 16 Form Detail Persimpangan

1.6 Pengujian Sistem

Pada tahapan ini, program secara keseluruhan akan diuji sebagai sistem yang lengkap untuk memastikan bahwa persyaratan perangkat lunak telah dipenuhi dan sesuai dengan kebutuhan serta keinginan pengguna. Pada penelitian ini teknik pengujian yang digunakan yaitu *Black Box* Testing, *White Box* Testing, dan Perbandingan antara hasil pencarian rute Algoritma PSO dan Algoritma Dijkstra.

1.6.1 Pengujian Black Box

Black Box Testing dilakukan dengan menjalankan sistem dan melakukan apa yang bisa dikerjakan oleh sistem untuk menguji tingkah laku dari sistem apakah keluaran yang dihasilkan sistem sudah sesuai dengan masukan yang diberikan. Sebelum melakukan pengujian, ditentukan terlebih dahulu skenario pengujian untuk memaparkan urutan yang akan diuji pada sistem ini. Berdasarkan tabel skenario black box testing yang telah dibentuk, tahap selanjutnya ialah melakukan pengujian sesuai dengan yang dijelaskan pada tabel 4.1. Hasil pengujian dari skenario pengujian tersebut adalah sebagai berikut.

Tabel 4. 1 Hasil Pengujian Black Box

Kelas Uji	Nama Kasus Uji	Identifikasi	Hasil Pengujian
Pengujian Antarmuka *User*	Pengujian Fungsi Menampilkan Rute Perjalanan Optimum dari titik awal ke akhir sesuai dengan inputan *User* pada Google Maps.	1-1	Berhasil
	Pengujian Fungsi Menampilkan Rute Perjalanan Optimum dari titik awal ke akhir sesuai dengan inputan *User* pada tampilan Infografis	1-2	Berhasil

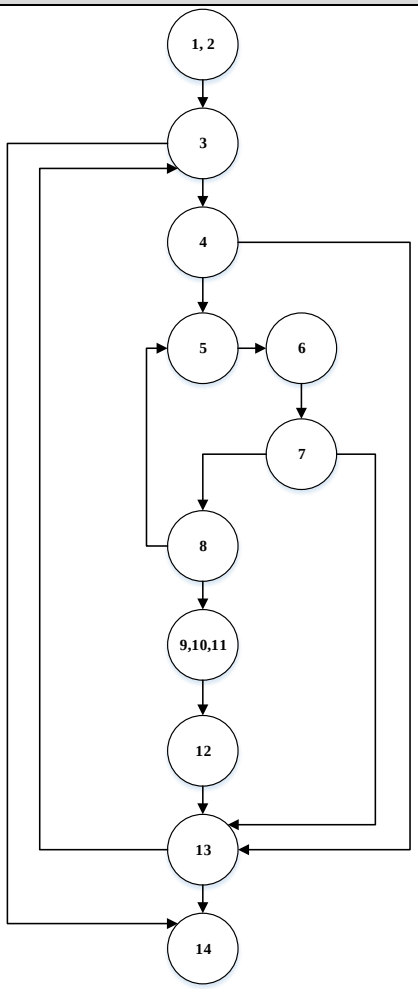
	Pengujian Fungsi Menampilkan seluruh Persimpangan (Node) pada Wilayah Denpasar Utara.	1-3	Berhasil
	Pengujian Fungsi Menampilkan Menu Detail Persimpangan	1-4	Berhasil
	Pengujian Fungsi Menampilkan Menu Bantuan	1-5	Berhasil
Pengujian Login	Pengujian Fungsi Validasi Login Admin	2-1	Berhasil
Pengujian Antarmuka *Admin*	Pengujian Fungsi Menampilan Rute Perjalanan Optimum menggunakan Algoritma Particle Swarm Optimization	3-1	Berhasil
	Menampilan Rute Perjalanan Optimum menggunakan Algoritma Particle Swarm Optimization pada Google Maps	3-2	Berhasil
	Menampilkan Tabel Perbandingan Hasil dari Algoritma PSO dengan Dijkstra	3-3	Berhasil
Pengujian Fungsi Manajemen Data Persimpangan (Node)	Pengujian Fungsi Tambah Data Persimpangan	4-1	Berhasil
	Pengujian Fungsi Edit Data Persimpangan	4-2	Berhasil
	Pengujian Fungsi Hapus Data Persimpangan	4-3	Berhasil

	Pengujian Fungsi Menampilkan Data Persimpangan	4-4	Berhasil
Pengujian Fungsi Manajemen Data Jalan	Pengujian Fungsi Tambah Data Jalan	4-1	Berhasil
	Pengujian Fungsi Edit Data Jalan	4-2	Berhasil
	Pengujian Fungsi Hapus Data Jalan	4-3	Berhasil
	Pengujian Fungsi Menampilkan Data Jalan	4-4	Berhasil
Pengujian Fungsi Manajemen Data Jalan	Pengujian Fungsi Tambah Data Wilayah	5-1	Berhasil
	Pengujian Fungsi Edit Data Wilayah	5-2	Berhasil
	Pengujian Fungsi Hapus Data Wilayah	5-3	Berhasil
	Pengujian Fungsi Menampilkan Data Wilayah	5-4	Berhasil

1.6.2 Pengujian White Box

Pengujian akan dilakukan menggunakan metode *white-box* menggunakan teknik *basis path testing*. Pengujian dengan *basis path* ini adalah teknik pengujian struktur kontrol untuk menjamin semua statemen dalam setiap jalur independen program dieksekusi minimal 1 kali. Perhitungan jalur independen dapat dilakukan melalui metrik *Cyclomatic Complexity*. Pengujian ini akan difokuskan pada proses di dalam *Pencarian Shortest Path Algoritma PSO*.

Tabel 4. 2 Flowgraph Sistem

No.	Keterangan	Flowgraph
1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12. 13. 14.	Input Persimpangan Awal, Akhir, dan Parameter Inisialisasi Partikel Do While $done != TRUE IF $epoch < $max_epoch Then Foreach partikel Hitung pBest (waktu tempuh) IF pBest < TARGET END FOREACH Bubble Sort partikel dari partikel dengan pBest terkecil ke terbesar Hitung Kecepatan (V) partikel Update partikel kecuali partikel ke-0 End IF Then End Do while Cetak pBest partikel ke-0 (gBest)	

Kompleksitas siklomatis adalah matriks perangkat lunak yang memberikan pengukuran kuantitatif terhadap kompleksitas logis suatu program. Bila matrik ini digunakan dalam konteks metode pengujian *basis path*, maka nilai yang terhitung untuk kompleksitas siklomatis menentukan jumlah jalur independen dalam basis

set suatu programan memberi batas atas bagi jumlah pengujian yang harus dilakukan untuk memastikan bahwa semua statemen telah dieksekusi sedikitnya satu kali.

Perhitungan nilai *cyclomatic complexity* alur proses pencarian rute optimum yang telah ditentukan pada tabel 4.26 adalah sebagai berikut.

V(G) = E – N + 2

= 15 – 14 + 2

= 3

Berdasarkan jumlah *cyclomatic complexity* maka dapat dibuat jalur (path) pengujian sesuai *flowgraph*

1. *Path 1*: 1-2-3-4-5-6-7-13-14
2. *Path 2*: 1-2-3-4-13-14
2. *Path 3*: 1-2-3-4-5-6-7-8-9-10-11-12-13-14

Nilai yang didapat adalah 3, dimana terdapat 3 jalur pengujian yang didapat berdasar perhitungan dari *flowgraph* tersebut. Nilai *cyclomatic complexity* yang dimiliki adalah 3 yang tergolong program sederhana, tanpa banyak resiko.

Tabel 4. 3 Kasus Uji

Path	***Input***	***Output yang diharapkan***	***Output***	***Keterangan***
1	Node Awal: Node 11 (Antasura - Antasura) Node Akhir : Node 30 (Mahendradata – Buluh Indah) Jumlah Partikel:1 Kecepatan Maksimum: 8 Iterasi Maksimum: 2	**11 - 10 - 15 - 16 - 20 - 63 - 66 - 43 - 32 - 30** **Bobot:** 11.16	**11 - 10 - 15 - 16 - 20 - 63 - 66 - 43 - 32 - 30** **Bobot:** 11.16	Sesuai

2	Node Awal: Node 11 (Antasura - Antasura) Node Akhir : Node 30 (Mahendradata – Buluh Indah) Jumlah Partikel:1 Kecepatan Maksimum: 8 Iterasi Maksimum: 1	**11 - 8 - 7 - 5 - 6 - 12 - 13 - 14 - 15 - 16 - 20 - 19 - 27 - 43 - 32 – 30** **Bobot:** 23.74	**11 - 8 - 7 - 5 - 6 - 12 - 13 - 14 - 15 - 16 - 20 - 19 - 27 - 43 - 32 – 30** **Bobot:** 23.74	Sesuai
3	Node Awal: Node 11 (Antasura - Antasura) Node Akhir : Node 30 (Mahendradata – Buluh Indah) Jumlah Partikel:2 Kecepatan Maksimum: 8 Iterasi Maksimum: 2	**11 - 8 - 7 - 10 - 15 - 16 - 20 - 23 - 26 - 28 – 30** **Bobot:** 13.77	**11 - 8 - 7 - 10 - 15 - 16 - 20 - 23 - 26 - 28 – 30** **Bobot:** 13.77	Sesuai

Pada tabel 4.3, terdapat 3 path yang akan diuji. Untuk menguji path-path yang didapatkan dari nilai cyclometic complexity sebelumnya, maka beberapa parameter harus ditetapkan agar urutan proses yang dijalankan sesuai dengan path yang ada. Untuk persimpangan awal dan akhir, serta kecepatan ditetapkan sama pada setiap path. Pada path 1, jumlah partikel ditetapkan menjadi 1 agar proses 5 pada Tabel 4.2 dapat dijalankan satu kali. Iterasi ditetapkan 2 agar proses perulangan 4 pada Tabel 4.2 dijalankan lebih dari satu kali. Pada path 2 jumlah partikel dan kecepatan maksimum ditetapkan bernilai 1 agar proses 5 dan 4 masing-masing dijalankan 1 kali. Pada Path 3, jumlah iterasi dan partikel ditetapkan 2 agar proses 5 dan 4 pada Tabel 4.2 dijalankan lebih dari 1 kali.

Output yang diharapkan merupakan hasil dari perhitungan manual. Kemudian akan dibandingkan dengan hasil sistem pada kolom Output yang merupakan keluaran dari sistem. Jika hasil sama maka, setiap proses pada path berhasil diuji.

1.6.3 Pengujian Perbandingan Hasil dengan Algoritma Dijkstra

Pengujian hasil yang dilakukan dengan memilih secara acak kombinasi node awal dan akhir dan kemudian akan dibandingkan hasil perhitungan antara Algoritma Particle Swarm Optimization dengan Algoritma Dijkstra. Parameter dari algoritma PSO yang digunakan adalah sebagai berikut:

Tabel 4. 4 Parameter Algoritma PSO (El-Sisi, 2014)

Kecepatan Maksimum	8
Jumlah Partikel	100
Jumlah Iterasi	100

Pengujian dilakukan sebanyak 100 kali percobaan. Rute perjalanan yang dihasilkan ditandai dengan urutan *Id* dari node atau persimpangan yang dilalui dimulai dari persimpangan awal dan berakhir di persimpangan akhir. Algoritma Dijkstra akan digunakan sebagai pembanding. Karena Algoritma Dijkstra merupakan algoritma yang popular dalam menyelesaikan permasalahan pencarian rute optimum dan mampu menjamin keoptimalannya (Dramski, Mariusz, 2012). Jika rute perjalanan yang dihasilkan sama atau waktu tempuh yang dihasilkan oleh Algoritma PSO sama dengan waktu tempuh dari Algoritma Dijkstra maka rute perjalanan berhasil menemukan rute terbaik. Hasil dari Algoritma Dijkstra juga akan digunakan sebagai bobot target dari perhitungan oleh Algoritma PSO.

Berikut adalah tabel hasil pengujian perbandingan yang dilakukan:

Tabel 4. 5 Hasil Perbandingan Hasil Antara PSO dan Dijkstra

No.	**Node Awal**	**Node Akhir**	**Rute PSO**	**Bobot PSO**	**Rute Dijkstra**	**Bobot Dijstra**	**Keterang-an**
1	67	49	67 68 72 75 79 55 49	5.1	67 68 72 75 79 55 49	5.1	Sesuai
2	42	46	42 43 66 67 68 72 75 79 55 49 48 47 46	11.16	42 - 43 - 66 - 67 - 68 - 72 - 75 - 79 - 80 - 81 - 86 - 87 - 90	11.05	Tidak Sesuai

					- 52 - 51 - 48 - 47 - 46		
3	48	80	48 51 52 90 89 88 87 86 81 80	4.15	48 51 52 90 89 88 87 86 81 80	4.15	Sesuai
4	8	46	8 11 45 46	4.99	8 11 45 46	4.99	Tidak Sesuai
5	48	17	48 49 11 10 15 16 17	10.49	48 49 11 10 15 16 17	10.49	Sesuai
6	22	70	22 24 25 27 43 66 67 68 72 71 70	8.16	22 24 25 27 43 66 67 68 72 71 70	8.16	Sesuai
7	87	103	87 86 81 80 79 78 100 99 98 97 96 95 103	6.61	87 86 81 80 79 78 100 99 98 97 96 95 103	6.61	Sesuai
8	69	24	69 68 67 66 43 27 25 24	6.54	69 68 67 66 43 27 25 24	6.54	
9	54	44	54 86 81 80 55 49 11 45 44	13.77	54 86 81 80 55 49 11 45 44	13.77	Sesuai
10	18	99	18 19 20 63 66 65 92 98 101 100 99	9.4	18 19 27 43 42 41 40 105 101 100 99	9.4	Sesuai
11	29	68	29 28 30 32 43 66 67 68	5.57	29 28 30 32 43 66 67 68	5.57	Sesuai
12	89	115	89 115	2.52	89 115	2.52	Sesuai
13	73	18	73 71 72 68 67 66 43 27 19 18	7.14	73 71 72 68 67 66 43 27 19 18	7.14	Sesuai
14	108	95	-108 - 109 - 84 - 82 - 81 - 80 - 79 - 78 - 100 -	9.590	-108 - 109 - 84 - 82 - 81 - 80 - 79 - 78 - 100 -	9.590	Sesuai

			99 - 98 - 97 - 96 - 95		99 - 98 - 97 - 96 - 95		
15	57	75	57 75	0.12	57 75	0.12	Sesuai
16	4	98	4 3 5 7 10 15 16 20 63 66 65 92 98	13.37	4 3 5 7 10 15 16 20 63 66 65 92 98	13.37	Sesuai
17	90	37	90 89 88 87 86 81 80 79 75 72 68 67 66 43 42 41 40 37	7.09	90 89 88 87 86 81 80 79 75 72 68 67 66 43 42 41 40 37	7.09	Sesuai
18	12	68	12 13 14 18 19 20 63 66 67 68	9.39	12 13 14 18 19 20 63 66 67 68	9.39	Sesuai
19	3	88	3 - 5 - 7 - 10 - 15 - 16 - 20 - 63 - 66 - 67 - 68 - 72 - 75 - 79 - 80 - 81 - 86 - 87 - 90 - 89 - 88	12.41	3 - 5 - 7 - 10 - 15 - 16 - 20 - 63 - 66 - 67 - 68 - 72 - 75 - 79 - 80 - 81 - 86 - 87 - 90 - 89 - 88	12.41	Sesuai
20	4	71	4 8 11 49 55 79 75 72 71	11.59	4 8 11 49 55 79 75 72 71	11.59	Sesuai
21	20	59	20 63 66 67 68 61 59	3.54	20 63 66 67 68 61 59	3.54	Sesuai
22	102	81	102 - 105 - 106 - 101 - 100 - 108 - 109 - 84 - 82 - 81	5.97	102 - 105 - 106 - 101 - 100 - 108 - 109 - 84 - 82 - 81	5.97	Sesuai
23	78	43	78 - 76 - 75 - 72 - 68 - 67 - 66 - 43	3.67	78 - 76 - 75 - 72 - 68 - 67 - 66 - 43	3.67	Sesuai
24	71	3	71 - 72 - 75 - 79 - 55 -	12.74	71 - 72 - 75 - 79 - 55 -	12.74	Sesuai

			49 - 11 - 8 - 4 - 3		49 - 11 - 8 - 4 - 3		
25	85	49	85 - 88 - 87 - 86 - 81 - 80 - 55 - 49	3.73	85 - 88 - 87 - 86 - 81 - 80 - 55 - 49	3.73	Sesuai
26	94	100	94 - 95 - 103 - 104 - 105 - 106 - 101 - 100	2.67	94 - 95 - 103 - 104 - 105 - 106 - 101 - 100	2.67	Sesuai
27	57	76	57 - 75 - 79 - 78 - 76	1.2	57 75 72 74 76	1.2	Sesuai
28	23	80	23 22 24 25 27 43 66 67 68 72 75 79 80	9.24	23 22 24 25 27 43 66 67 68 72 75 79 80	9.24	Sesuai
29	25	62	25 27 43 66 67 62	4.5	25 27 43 66 67 62	4.5	Sesuai
30	67f	83	67 66 65 92 98 101 100 108 109 84 83	8.27	67 66 65 92 98 101 100 108 109 84 83	8.27	Sesuai
31	113	6	113 85 88 87 86 81 80 55 49 11 10 6	14.61	113 85 88 87 86 81 80 55 49 11 10 6	14.61	Sesuai
32	32	108	32 - 43 - 42 - 41 - 40 - 37 - 93 - 92 - 98 - 101 - 100 - 108	7.32	32 - 43 - 42 - 41 - 40 - 37 - 93 - 92 - 98 - 101 - 100 - 108	7.32	Sesuai
33	27	85	27 - 43 - 66 - 67 - 68 - 72 - 75 - 79 - 80 - 81 - 86 - 87 - 90	6.31	27 - 43 - 66 - 67 - 68 - 72 - 75 - 79 - 80 - 81 - 86 - 87 - 90	6.31	Sesuai

			- 89 - 88 - 85		- 89 - 88 - 85		
34	75	66	75 72 68 67 66	1.86	75 72 68 67 66	1.86	Sesuai
35	45	15	45 11 10 15	5.93	45 11 10 15	5.93	Sesuai
36	30	98	30 - 32 - 43 - 42 - 41 - 40 - 37 - 93 - 92 - 98	6.37	30 32 43 42 41 40 37 93 92 98	6.37	Sesuai
37	101	56	101 - 100 - 108 - 109 - 84 - 82 - 81 - 80 - 79 - 75 - 57 - 56	6.88	101 - 100 - 108 - 109 - 84 - 82 - 81 - 80 - 79 - 75 - 57 - 56	6.88	Sesuai
38	17	90	17 16 20 63 66 67 68 72 75 79 80 81 86 87 90	7.06	17 16 20 63 66 67 68 72 75 79 80 81 86 87 90	7.06	Sesuai
39	108	87	108 - 109 - 111 - 112 - 114 - 113 - 85 - 88 - 87	3.5	108 - 109 - 111 - 112 - 114 - 113 - 85 - 88 - 87	3.5	Sesuai
40	86	22	86 - 81 - 80 - 79 - 75 - 72 - 68 - 67 - 66 - 43 - 27 - 25 - 24 - 22	8.64	86 - 81 - 80 - 79 - 75 - 72 - 68 - 67 - 66 - 43 - 27 - 25 - 24 - 22	8.64	Sesuai
41	80	82	80 81 86 87 90 89 88 85 82	1.8	80 81 86 87 90 89 88 85 82	1.8	Sesuai
42	36	71	36 - 35 - 38 - 39 - 41 - 42 - 43 - 66 - 67 - 68 - 72 - 71	6.42	36 - 35 - 38 - 39 - 41 - 42 - 43 - 66 - 67 - 68 - 72 - 71	6.42	Sesuai

43	97	4	97 - 96 - 102 - 105 - 106 - 101 - 100 - 108 - 109 - 84 - 82 - 81 - 80 - 55 - 49 - 11 - 8 - 4 -	16.69	97 - 96 - 102 - 105 - 106 - 101 - 100 - 108 - 109 - 84 - 82 - 81 - 80 - 55 - 49 - 11 - 8 - 4 -	16.69	Sesuai
44	78	97	78 - 100 - 99 - 98 - 97	4.95	78 - 100 - 99 - 98 - 97	4.95	Sesuai
45	26	35	26 28 30 32 43 42 41 40 37 36 35	8.57	26 28 30 32 43 42 41 40 37 36 35	8.57	Sesuai
46	69	71	69 70 71	0.84	69 70 71	0.84	Sesuai
47	22	67	22 - 24 - 25 - 27 - 43 - 66 - 67	6.18	22 - 24 - 25 - 27 - 43 - 66 - 67	6.18	Sesuai
48	115	106	115 114 113 85 82 81 80 79 78 100 99 98 97 96 102 105 106	9.55	115 114 113 85 88 87 86 81 80 79 75 57 58 60 102 105 106	9.51	Tidak Sesuai
49	46	87	46 47 48 51 52 90 89 88 87	7.2	46 47 48 51 52 90 89 88 87	7.2	Sesuai
50	33	6	33 32 43 27 19 18 14 13 12 6	11.76	33 32 43 27 19 18 14 13 12 6	11.76	Sesuai
51	67	103	67 - 66 - 65 - 92 - 98 - 97 - 96 - 95 - 103 -	5.19	67 - 66 - 65 - 92 - 98 - 97 - 96 - 95 - 103 -	5.19	Sesuai
52	26	89	26 - 28 - 30 - 32 - 43 - 66 - 67 - 68 - 72 - 75 -	8.64	26 - 28 - 30 - 32 - 43 - 66 - 67 - 68 - 72 - 75 -	8.64	Sesuai

			79 - 80 - 81 - 86 - 87 - 90 - 89		79 - 80 - 81 - 86 - 87 - 90 - 89		
53	34	74	34 - 30 - 32 - 43 - 66 - 67 - 68 - 72 - 71 - 73 - 74	8.7	34 - 30 - 32 - 43 - 66 - 67 - 68 - 72 - 71 - 73 - 74	8.7	Sesuai
54	5	71	5 7 10 15 16 20 63 66 67 68 72 71	9.87	5 7 10 15 16 20 63 66 67 68 72 71	9.87	Sesuai
55	74	91	74 76 75 79 80 81 86 87 90 52 91	2.78	74 76 75 79 80 81 86 87 90 52 91	2.78	Sesuai
56	56	66	56 57 75 72 68 67 66	2.88	56 57 75 72 68 67 66	2.88	Sesuai
57	92	63	92 - 98 - 97 - 93 - 37 - 38 - 39 - 41 - 42 - 43 - 66 - 63 -	6.21	92 - 98 - 97 - 93 - 37 - 38 - 39 - 41 - 42 - 43 - 66 - 63 -	6.21	Sesuai
58	113	103	113 - 85 - 88 - 87 - 86 - 81 - 80 - 79 - 78 - 100 - 99 - 98 - 97 - 96 - 95 - 103	8.59	113 - 85 - 88 - 87 - 86 - 81 - 80 - 79 - 78 - 100 - 99 - 98 - 97 - 96 - 95 - 103	8.59	Sesuai
59	99	50	99 - 98 - 101 - 100 - 108 - 109 - 111 - 112 - 114 - 113 - 85 - 88 - 87 - 90 - 52 - 51 - 50	9.55	99 - 98 - 101 - 100 - 108 - 109 - 111 - 112 - 114 - 113 - 85 - 88 - 87 - 90 - 52 - 51 - 50	9.55	Sesuai

60	24	18	24 25 18	1.2	24 25 18	1.2	Sesuai
61	99	102	99 - 98 - 97 - 96 - 102 -	1.53	99 - 98 - 97 - 96 - 102 -	1.53	Sesuai
62	24	18	24 25 18	1.2	24 25 18	1.2	Sesuai
63	99	102	99 98 97 90 89 88 87 86 81 80 79 75 57 58 60 102	4.54	99 98 97 90 89 88 87 86 81 80 79 75 57 58 60 102	4.54	Sesuai
64	21	66	21 - 22 - 24 - 25 - 27 - 43 - 66 -	6.78	21 - 22 - 24 - 25 - 27 - 43 - 66 -	6.78	Sesuai
65	75	19	75 72 68 67 66 43 27 19	5.46	75 72 68 67 66 43 27 19	5.46	Sesuai
66	24	15	24 25 18 14 15	4.83	24 25 18 14 15	4.83	Sesuai
67	102	82	102 105 106 101 100 108 109 84 82	5.79	102 105 106 101 100 108 109 84 82	5.79	Sesuai
68	31	59	31 32 43 66 67 68 61 59	4.08	31 32 43 66 67 68 61 59	4.08	Sesuai
69	36	31	36 35 38 39 41 42 43 32 31	5.34	36 35 38 39 41 42 43 32 31	5.34	Sesuai
70	105	37	105 106 97 93 37	1.78	105 106 97 93 37	1.78	Sesuai
71	47	13	47 46 45 11 10 15 14 13	12.95	47 46 45 11 10 15 14 13	12.95	Sesuai
72	11	104	11 49 55 79 78 100 99 98 97 96 95 103 104	13.12	11 49 55 79 78 100 99 98 97 96 95 103 104	13.12	Sesuai
72	30	21	30 28 26 24 22 21	6.11	30 28 26 24 22 21	6.11	Sesuai

73	95	48	95 - 103 - 104 - 105 - 106 - 101 - 100 - 108 - 109 - 84 - 82 - 81 - 86 - 87 - 90 - 52 - 51 - 48	10.09	95 - 103 - 104 - 105 - 106 - 101 - 100 - 108 - 109 - 111 - 112 - 114 - 113 - 85 - 88 - 87 - 90 - 52 - 51 - 48 -	9.9	Tidak Sesuai
74	73	25	73 71 72 68 67 66 43 27 25	6.66	73 71 72 68 67 66 43 27 25	6.66	Sesuai
75	49	59	49 55 79 75 57 59	4.14	49 55 79 75 57 59	4.14	Sesuai
76	77	66	77 76 75 72 68 67 66	2.47	77 76 75 72 68 67 66	2.47	Sesuai
77	43	90	43 66 67 68 72 75 79 80 81 86 87 90	3.79	43 66 67 68 72 75 79 80 81 86 87 90	3.79	Sesuai
78	75	44	75 79 55 49 11 45 44	13.47	75 79 55 49 11 45 44	13.47	Sesuai
79	99	44	99 98 101 100 108 109 84 82 81 80 55 49 11 45 44	19.36	99 98 101 100 108 109 84 82 81 80 55 49 11 45 44	19.36	Sesuai
80	95	15	95 103 104 105 106 97 93 37 38 39 41 42 43 27 19 20 16 15	12.2	95 103 104 105 106 97 93 37 38 39 41 42 43 27 19 20 16 15	12.2	Sesuai
81	24	84	24 25 27 43 42 41 40 37 93 92 98	11.63	24 25 27 43 42 41 40 37 93 92 98	11.63	Sesuai

			101 100 108 109 84		101 100 108 109 84		
82	21	15	21 22 24 25 18 14 15	7.05	21 22 24 25 18 14 15	7.05	Sesuai
83	83	102	83 84 82 81 80 79 78 100 99 98 97 96 102	8.92	83 84 82 81 80 79 78 100 99 98 97 96 102	8.92	Sesuai
84	27	43	27 43	1.8	27 43	1.8	Sesuai
85	39	113	39 40 37 93 92 98 101 100 108 109 111 112 114 113	5.6	39 40 37 93 92 98 101 100 108 109 111 112 114 113	5.6	Sesuai
86	112	99	112 114 113 85 88 87 86 81 80 79 78 100 99	7.73	112 114 113 85 88 87 86 81 80 79 78 100 99	7.73	Sesuai
87	86	26	86 81 80 79 75 72 71 70 69 68 67 66 43 32 30 28 26	10.19	86 - 81 - 80 - 79 - 75 - 72 - 68 - 67 - 66 - 43 - 32 - 30 - 28 - 26 -	8.34	Tidak Sesuai
88	9	53	9 4 8 11 49 48 51 52 53	12.01	9 4 8 11 49 48 51 52 53	12.01	Sesuai
89	83	110	83 84 82 81 80 79 78 100 108 109 110	9.55	83 84 82 81 80 79 78 100 108 109 110	9.55	Sesuai
90	65	53	65 42 43 66 67 68 72 75 79 80 81 86 87 90 52 53	6.37	65 42 43 66 67 68 72 75 79 80 81 86 87 90 52 53	6.37	Sesuai

91	88	108	88 87 86 81 80 79 78 100 108	5.45	88 87 86 81 80 79 78 100 108	5.45	Sesuai
92	88	38	88 87 86 81 80 79 75 72 68 67 66 65 42 41 40 37 38	7.51	88 - 87 - 86 - 81 - 80 - 79 - 75 - 72 - 68 - 67 - 66 - 43 - 42 - 41 - 40 - 37 - 38	6.97	Tidak Sesuai
93	84	12	84 82 81 80 79 75 72 68 67 66 43 27 19 18 14 13 12	13.38	84 82 81 80 79 75 72 68 67 66 43 27 19 18 14 13 12	13.38	Sesuai
94	17	72	17 16 20 63 66 67 68 72	5.31	17 16 20 63 66 67 68 72	5.31	Sesuai
95	86	15	86 81 80 79 75 72 68 67 66 43 27 19 20 16 15	9.78	86 81 80 79 75 72 68 67 66 43 27 19 20 16 15	9.78	Sesuai
96	67	100	67 66 65 92 98 101 100	5.17	67 66 65 92 98 101 100	5.17	Sesuai
97	20	69	20 63 66 67 68 69	3.72	20 63 66 67 68 69	3.72	Sesuai
98	109	44	109 84 82 81 80 55 49 11 8 4	13.37	109 84 82 81 80 55 49 11 8 4	13.37	Sesuai
99	70	61	70 69 68 61	1.92	70 71 72 68 61	1.62	Tidak Sesuai
100	75	78	75 79 78	0.54	75 79 78	0.54	Sesuai

Pada tabel 4.5, jika rute yang dihasilkan Algoritma PSO tidak sama atau waktu tempuh yang dihasilkan lebih lama daripada yang dihasilkan oleh Algoritma Dijkstra, maka pada kolom keterangan ditetapkan tidak sesuai.

Untuk mendapatkan hasil perbandingan dari Algortima PSO yang digunakan digunakan rumus:

$$Hasil\ Perbandingan = \frac{hasil\ yang\ sesuai}{banyak\ percobaan} \times 100\%$$

Jadi hasil perbandingan dengan Algoritma Dijkstra yang didapatkan adalah:

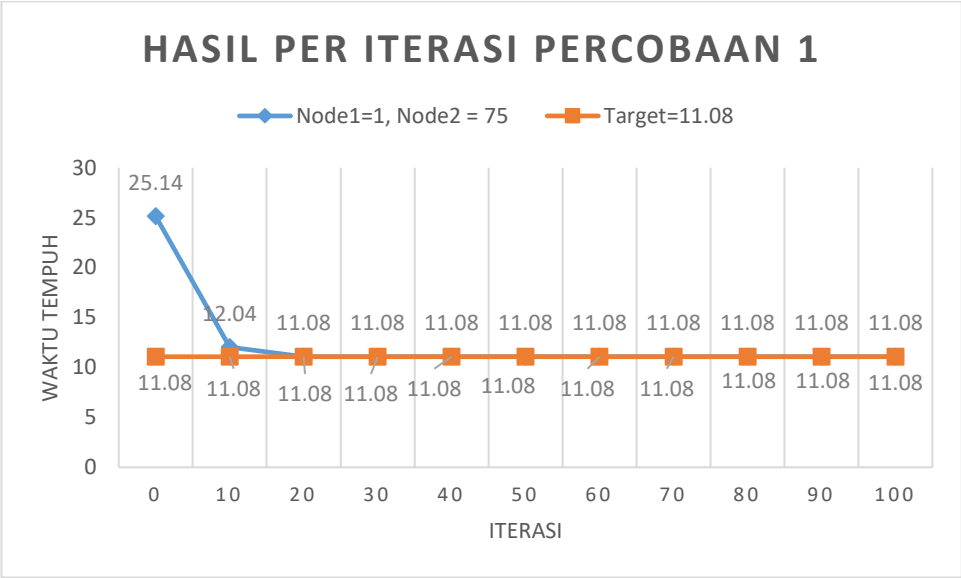
$$Hasil\ Perbandingan = \frac{93}{100} \text{ X } 100\% = 93\ \%$$

Untuk menguji kemampuan algoritma PSO dalam menemukan solusi per iterasinya, dilakukan pengujian dengan persimpangan awal (Node1) dan persimpangan akhir (Node2) yang berbeda. Akan dicatat solusi per iterasi dan dibandingkan apakah semakin bertambahnya jumlah iterasi semakin mendekati hasil dari Algoritma Dijkstra. Berikut adalah hasil perbandingnya.

Tabel 4. 6 Hasil Perbandingan Solusi per Iterasi

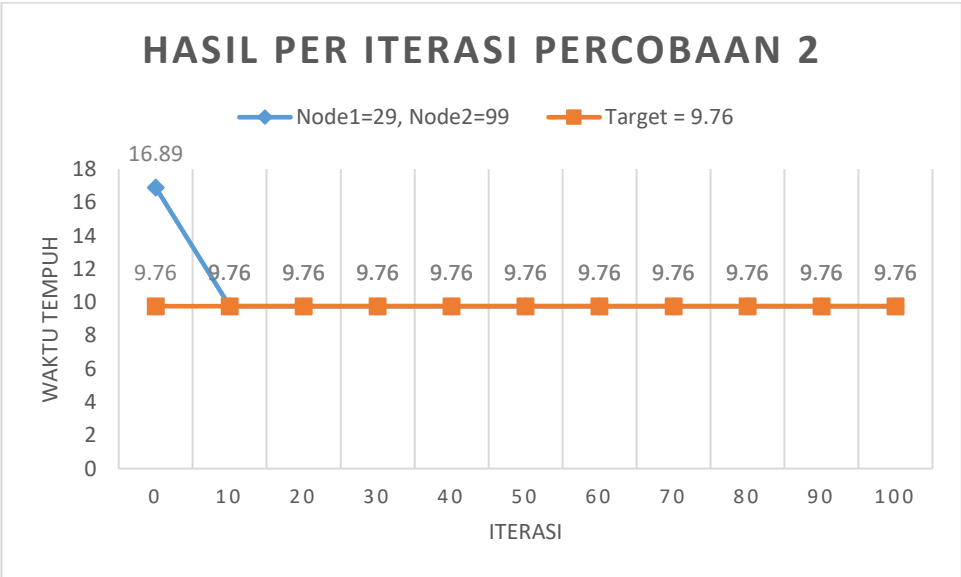
Iterasi	**Percobaan**			
	1	**2**	**3**	**4**
	Node1 = Ahmad Yani	**Node1 = Angsoka**	**Node1 = Antasura - Antasura**	**Node1 = Sari Gading - Ratna**
	Node2= Gatot Subroto I – Gatot Subroto	**Node2 = Jl. Yudistira**	**Node2 = Sri Rama - Sutomo**	**Node2 = Sentanu – A.Yani**
0	25.14	16.89	27.51	8.42
10	12.04	9.76	11.27	7.58
20	11.08	9.76	11.27	7.58
30	11.08	9.76	11.27	7.58
40	11.08	9.76	11.27	7.58
50	11.08	9.76	11.27	7.58
60	11.08	9.76	11.27	7.58
70	11.08	9.76	11.27	7.58
80	11.08	9.76	11.27	7.58
90	11.08	9.76	11.27	7.58
100	11.08	9.76	11.27	7.58
Hasil Dijkstra	11.08	9.76	11.27	7.58

Berikut merupakan grafik hasil perbandingan Algoritma Particle Swarm Optimization dan Algoritma Dijkstra yang sama.



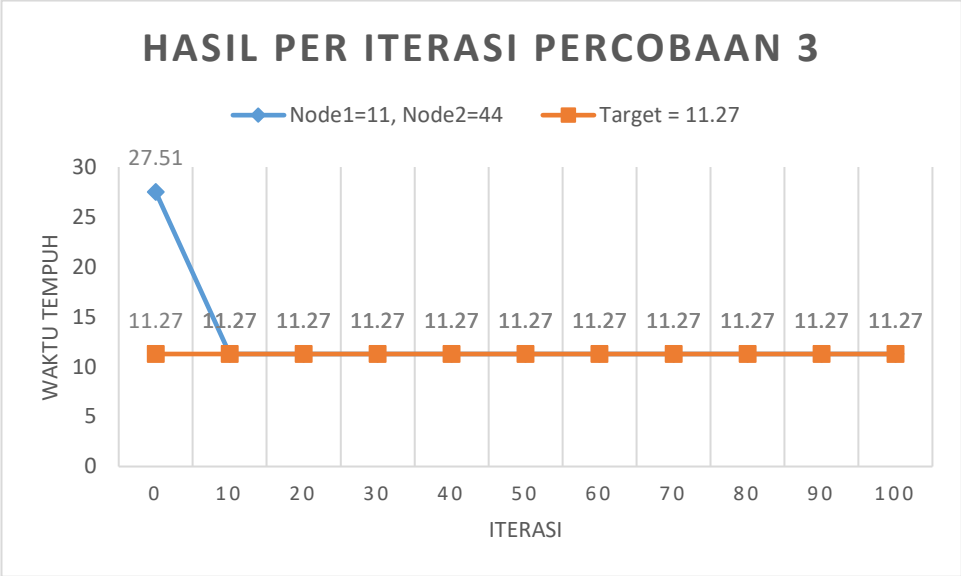
Gambar 4. 17 Grafik Hasil Per Iterasi Percobaan 1

Pada percobaan 1 Gambar 4.17 dengan Node1 = *Ahmad Yani* dan Node2 = *Gatot Subroto I – Gatot Subroto* pada iterasi ke-20 sudah memperoleh hasil yang sama dengan Algoritma *Dijkstra*



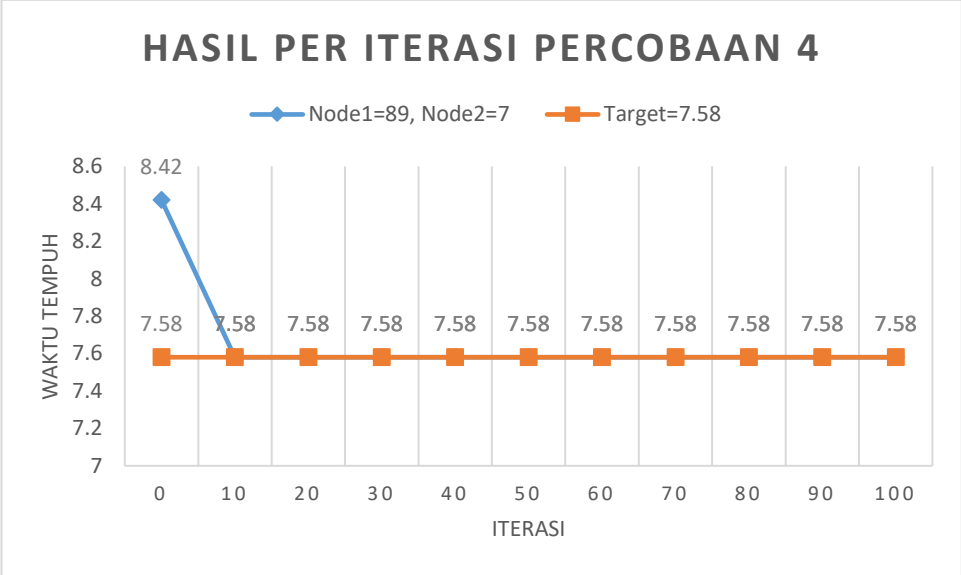
Gambar 4. 18 Grafik Hasil Per Iterasi Percobaan 1

Pada percobaan 2 Gambar 4.18 dengan Node1 = *Angsoka* dan Node2 = *Jl. Yudistira* pada iterasi ke-10 sudah memperoleh hasil yang sama dengan Algoritma *Dijkstra*.



Gambar 4. 19 Grafik Hasil Per Iterasi Percobaan 1

Pada percobaan 3 Gambar 4.19 dengan Node1 = *Antasura - Antasura* dan Node2 = *Sri Rama - Sutomo* pada iterasi ke-10 sudah memperoleh hasil yang sama dengan Algoritma *Dijkstra*.



Gambar 4. 20 Grafik Hasil Per Iterasi Percobaan 1

Pada percobaan 4 Gambar 4.20 dengan Node1 = *Sari Gading - Ratna* dan Node2 = *Sentanu – A.Yani* dengan target dari Algoritma Dijsktra sebesar 7.58, pada iterasi ke-10 sudah memperoleh hasil yang sama dengan Algoritma *Dijkstra*.

Pada beberapa percobaan, hasil dari Algoritma Particle Swarm Optimization tidak mendapatkan hasil yang sama dengan Algoritma Dijkstra sampai batas iterasi yang ditentukan yaitu 100 iterasi. Pada beberapa percobaan yaitu solusi partikel terjebak pada *local optimum*, dan pada beberapa percobaan pada iterasi ke-10 mulai mengalami penurunan waktu tempuh. Jika jumlah partikel dan jumlah iterasi ditambah melebihi dari Tabel 4.4, akan menyebabkan eksekusi program lebih dari 1 menit.

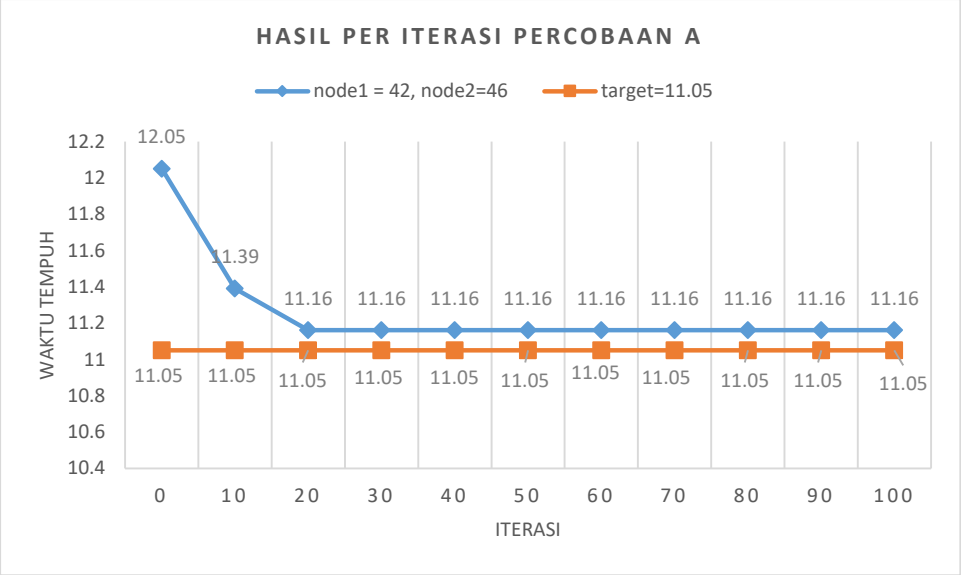
Berikut ini merupakan hasil dari Algoritma PSO yang tidak mendapatkan hasil yang sama dengan *Algoritma Dijkstra.*

Tabel 4. 7 Hasil Algoritma per Iterasi

Iterasi	**A**	**B**	**C**	**D**
	Node1= Maruti Cokroaminoto	**Node1= Kembang Kertas – Sari Gading**	**Node1= Sari Gading - Suli**	**Node1=Arjuna – Gajah Mada**
	Node2=Pertulaka - Cekomaria	**Node2= Gambuh - Setiabudi**	**Node2=Karya Makmur – Cargo Permai**	**Node2= Seroja - Padma**
0	14.16	9.23	21.2	13.93
10	11.16	7.51	10.68	10.09
20	11.16	7.51	10.68	10.09
30	11.16	7.51	10.68	10.09
40	11.16	7.51	10.61	10.09
50	11.16	7.51	10.61	10.09
60	11.16	7.51	10.61	10.09
70	11.16	7.51	10.61	10.09
80	11.16	7.51	10.61	10.09
90	11.16	7.51	10.61	10.09
100	11.16	7.51	10.61	10.09
Hasil Dijkstra	**11.05**	**6.97**	**8.03**	**9.9**

Pada Tabel 4.7 Percobaan A, dengan target 11.05 menit terjebak pada *local optimum* dimana pada iterasi ke 0 solusi sebesar 11.16 menit . Pada Percobaan B, dengan target 6.97 menit terjebak pada *local optimum* dimana pada iterasi ke-0 solusi sebesar 9.23 menit tetapi pada iterasi ke-10 solusi menjadi 7.51 menit . Pada Percobaan C, dengan target 8.03 menit terjebak pada *local optimum* dimana padaiterasi ke 10 solusi sebesar 10.68 menit sampai pada iterasi ke-100. Pada Percobaan D, dengan target 9.9 menit terjebak pada *local optimum* dimana pada iterasi ke 0 solusi sebesar 13.93 menit. Pada iterasi ke-10 mengalami penurunan waktu tempuh menjadi 10.09 menit dan sampai iterasi ke-100 solusi tetap pada 10.09 menit.

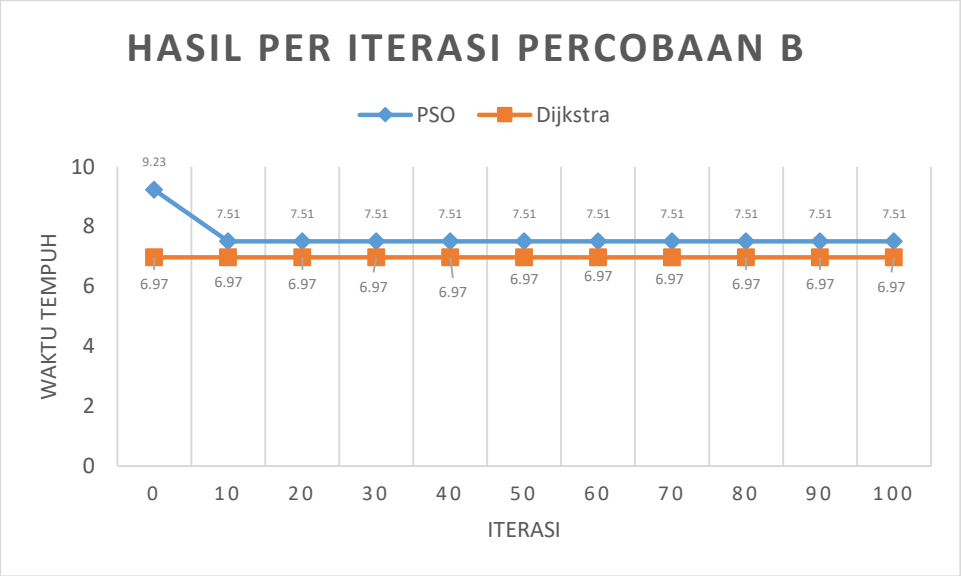
Berikut adalah grafik yang menunjukan solusi per iterasi dari Algoritma Particle Swarm Optimization. Garis biru merupakan solusi dari PSO sedangkan garis merah menunjukan solusi dari Algoritma Dijkstra yang merupakan target.



Gambar 4. 21 Hasil per Iterasi Percobaan A

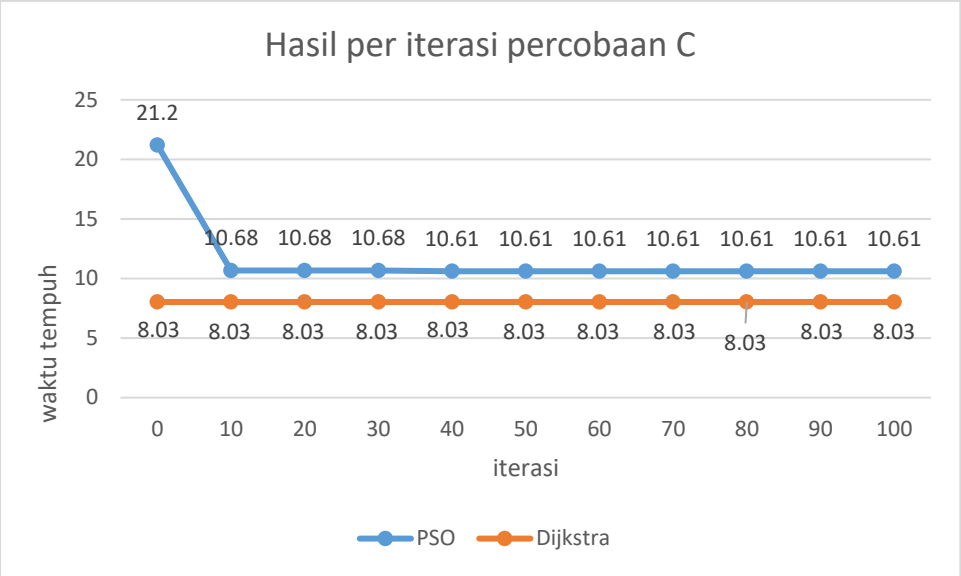
Pada Gambar 4.21 merupakan grafik untuk memperlihatkan hasil waktu tempuh yang dihasilkan oleh Algoritma PSO per iterasinya kemudian membandingkanya dengan hasi dari Algoritma Dijkstra. Jika waktu tempuh yang dihasilkan oleh sama, maka garis grafik akan beririsan. Sumbu absis menunjukan iterasi dan sumbu ordinat menunjukan waktu tempuh. Garis merah menjukan hasil

dari Algoritma Dijkstra, sedangkan garis biru menunjukan hasil dari Algoritma PSO. Pada Gambar 4.17 menunjukan hasil dari Algoritma PSO mengalami perubahan yaitu pada iterasi ke-10 dari waktu tempuh sebesar 12.05 menit menjadi 11.39 menit, dan pada iterasi ke-20 turun menjadi 11.16 menit dan sampai pada ke-100 nilai dari Algoritma PSO tetap yaitu 11.16 dan tidak beririsan dengan garis merah yang merupakan hasil dari Algoritma Dijkstra sebesar 11.05 menit.



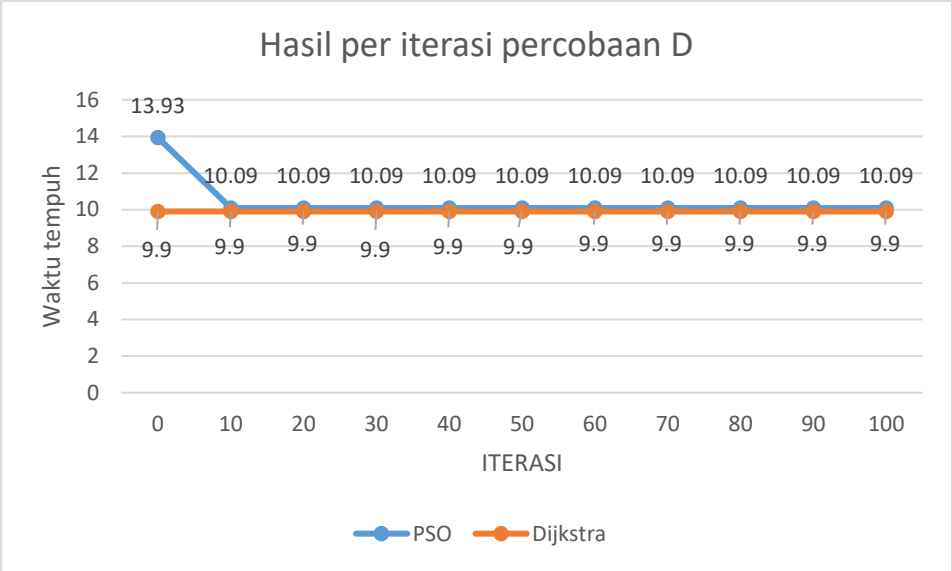
Gambar 4. 22 Hasil Perbandingan Solusi per Iterasi Percobaan B

Pada Gambar 4.22 merupakan grafik untuk memperlihatkan hasil waktu tempuh yang dihasilkan oleh Algoritma PSO per iterasinya kemudian membandingkanya dengan hasi dari Algoritma Dijkstra. Jika waktu tempuh yang dihasilkan oleh sama, maka garis grafik akan beririsan. Sumbu absis menunjukan iterasi dan sumbu ordinat menunjukan waktu tempuh. Garis merah menjukan hasil dari Algoritma Dijkstra, sedangkan garis biru menunjukan hasil dari Algoritma PSO. Pada Gambar 4.18 menunjukan hasil dari Algoritma PSO mengalami perubahan yaitu pada iterasi ke-10 dari waktu tempuh sebesar 9.23 menit menjadi 7.51 menit. Sampai pada iterasi ke-100 tetap sebesar 7.51 menit dan tidak beririsan dengan garis merah yang merupakan hasil dari Algoritma Dijkstra sebesar 6.93.



Gambar 4. 23 Hasil Perbandingan Solusi per Iterasi Percobaan C

Pada Gambar 4.23 merupakan grafik untuk memperlihatkan hasil waktu tempuh yang dihasilkan oleh Algoritma PSO per iterasinya kemudian membandingkanya dengan hasi dari Algoritma Dijkstra. Jika waktu tempuh yang dihasilkan oleh sama, maka garis grafik akan beririsan. Sumbu absis menunjukan iterasi dan sumbu ordinat menunjukan waktu tempuh. Garis merah menjukan hasil dari Algoritma Dijkstra, sedangkan garis biru menunjukan hasil dari Algoritma PSO. Pada Gambar 4.18 menunjukan hasil dari Algoritma PSO mengalami perubahan yaitu pada iterasi ke-10 dari waktu tempuh sebesar 21.2 menit menjadi 10.68 menit. Sampai pada iterasi ke-100 tetap sebesar 10.68 menit dan tidak beririsan dengan garis merah yang merupakan hasil dari Algoritma Dijkstra sebesar 8.03 menit.



Gambar 4. 24 Hasil Perbandingan Solusi per Iterasi Percobaan D

Pada Gambar 4.24 merupakan grafik untuk memperlihatkan hasil waktu tempuh yang dihasilkan oleh Algoritma PSO per iterasinya kemudian membandingkanya dengan hasi dari Algoritma Dijkstra. Jika waktu tempuh yang dihasilkan oleh sama, maka garis grafik akan beririsan. Sumbu absis menunjukan iterasi dan sumbu ordinat menunjukan waktu tempuh. Garis merah menjukan hasil dari Algoritma Dijkstra, sedangkan garis biru menunjukan hasil dari Algoritma PSO. Pada Gambar 4.18 menunjukan hasil dari Algoritma PSO mengalami perubahan yaitu pada iterasi ke-10 dari waktu tempuh sebesar 13.93 menit menjadi 10.09 menit. Sampai pada iterasi ke-100 tetap sebesar 10.68 menit dan tidak beririsan dengan garis merah yang merupakan hasil dari Algoritma Dijkstra yaitu sebesar 9.9 menit.