BAB 4
IMPLEMENTASI DAN PENGUJIAN SISTEM

4.1 Implementasi Sistem

Tahap implementasi sistem merupakan tahap pembuatan perangkat lunak, tahap lanjut dari tahap perancangan sistem. Tahap yang dilakukan untuk menerjemahkan perancangan berdasarkan hasil analisis dalam bahasa yang dimengerti oleh komputer serta penerapan perangkat lunak pada keadaan yang sebenarnya.

4.1.1 Implementasi Perangkat Keras

Spesifikasi perangkat keras yang digunakan untuk melakukan analisis perbandingan algoritma *ant colony optimization system* dengan algoritma genetika untuk mencari langkah optimal dalam penyelesaian permainan Sudoku adalah sebagai berikut :

Tabel 4.1 Spesifikasi Perangkat Keras

No	Perangkat Keras	Spesifikasi
1	*Processor*	Intel Core i3-2310M 2.10GHz
2	RAM	4GB
3	VGA	1GB
4	*Harddisk*	500GB
5	Layar	1366x768 *pixels*

4.1.2 Implementasi Perangkat Lunak

Spesifikasi perangkat lunak yang digunakan untuk melakukan analisis perbandingan algoritma *ant colony optimization system* dengan algoritma genetika untuk mencari langkah optimal dalam penyelesaian permainan Sudoku adalah sebagai berikut :

Tabel 4.2 Spesifikasi Perangkat Lunak

No	Perangkat Lunak	Spesifikasi
1	Sistem Operasi	*Windows* 7
2	Bahasa Pemrograman	C# (C *Sharp*)
3	*Tool* Pembangun	*Microsoft Visual Studio* 2010

4.1.3 Implementasi Antarmuka

Implementasi antarmuka dibangun berdasarkan perancangan yang telah dilakukan sebelumnya. Implementasi antamuka pada analisis perbandingan algoritma *ant colony optimization system* dengan algoritma genetika untuk mencari langkah optimal dalam penyelesaian permainan Sudoku adalah sebagai berikut:

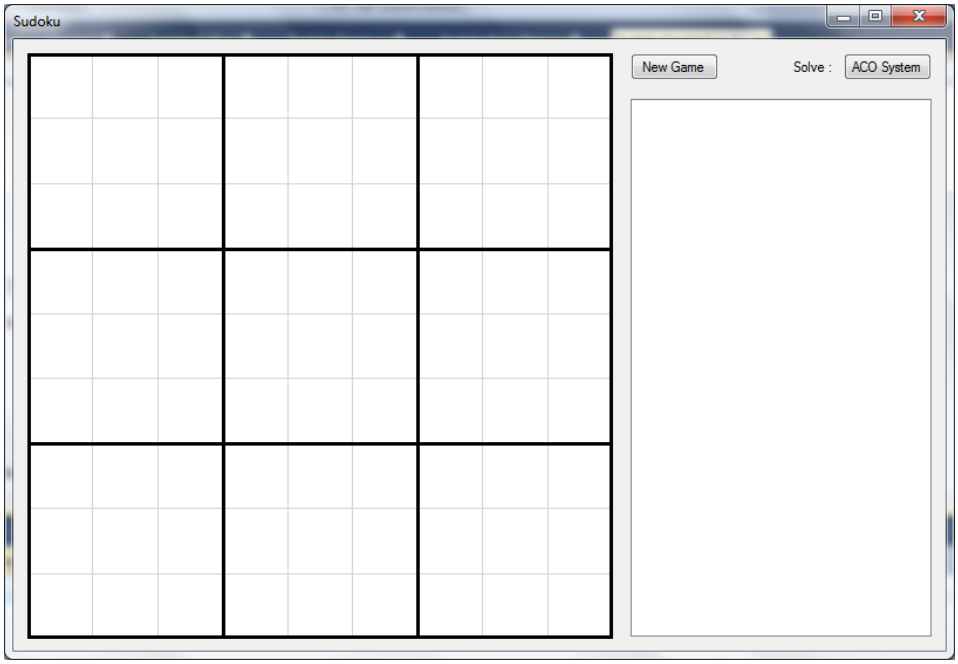
Tabel 4.3 Implementasi Antarmuka

No	Antarmuka	*Use Case*	Nama File
1	Utama		Program.cs
2	*New Game*	*New Game*	Form1.cs
			SudokuGame.cs
			SudokuCell.cs
			SudokuCellGroup.cs
			SudokuLine.cs
			SudokuColumn.cs
			SudokuSquare.cs
			SudokuControl.cs
3	*ACO System*	Algoritma *ACO System*	Form1.cs
			SudokuGame.cs
			SudokuSolveStep.cs
			SudokuCell.cs
			SudokuCellGroup.cs

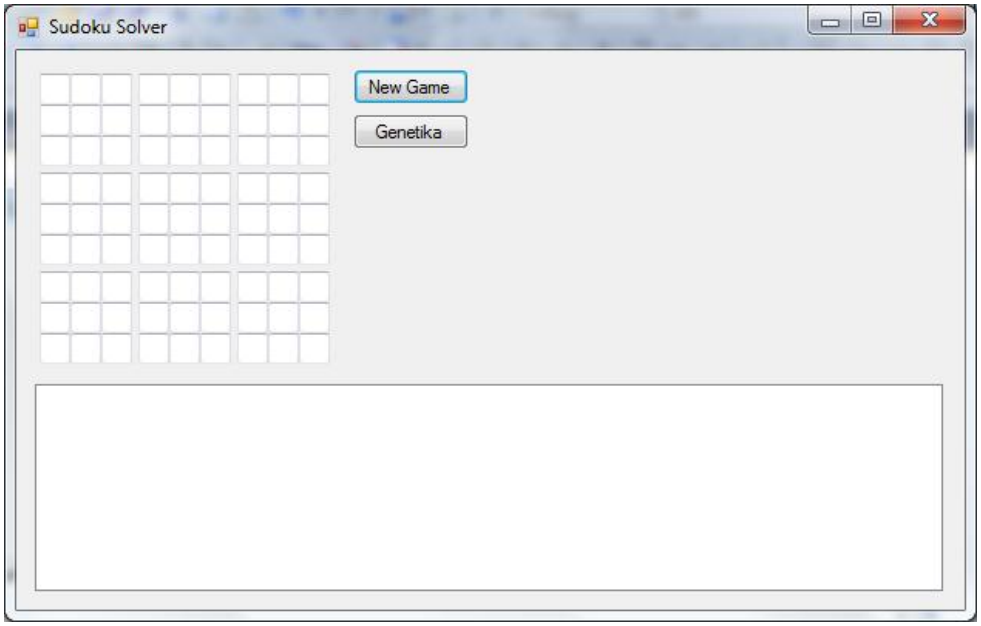
			SudokuLine.cs
			SudokuColumn.cs
			SudokuSquare.cs
			SudokuControl.cs
4	Genetika	Algoritma Genetika	Form1.vb

1. Antarmuka Utama

   Antarmuka ini merupakan tampilan awal saat aplikasi dijalankan.



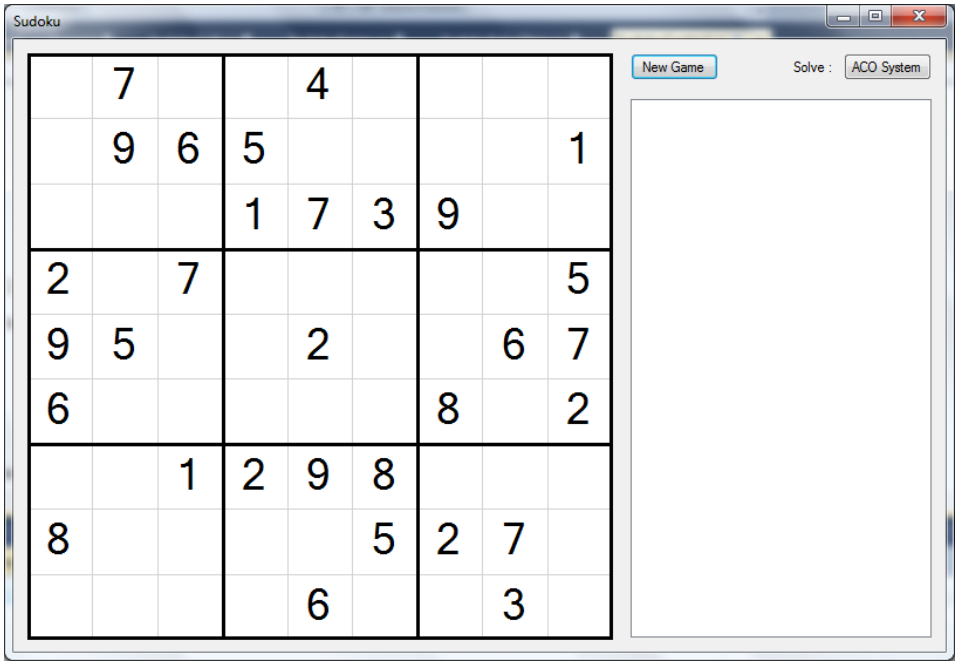
Gambar 4.1 Antarmuka Utama (*ACO System*)



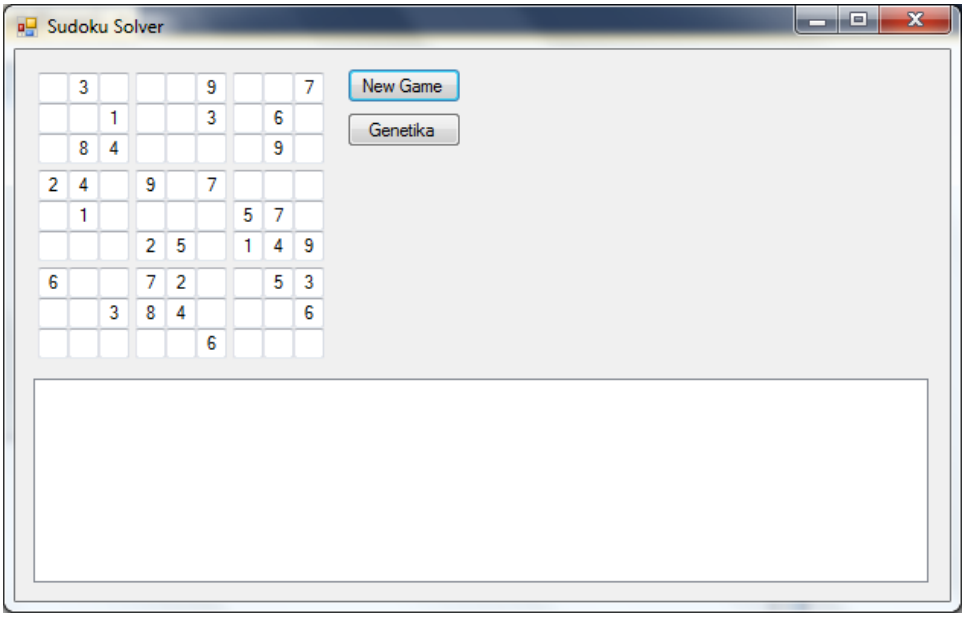
Gambar 4.2 Antarmuka Utama (Genetika)

2. Antarmuka *New Game*

   Antarmuka ini merupakan tampilan setelah pengguna menekan tombol *New Game*.



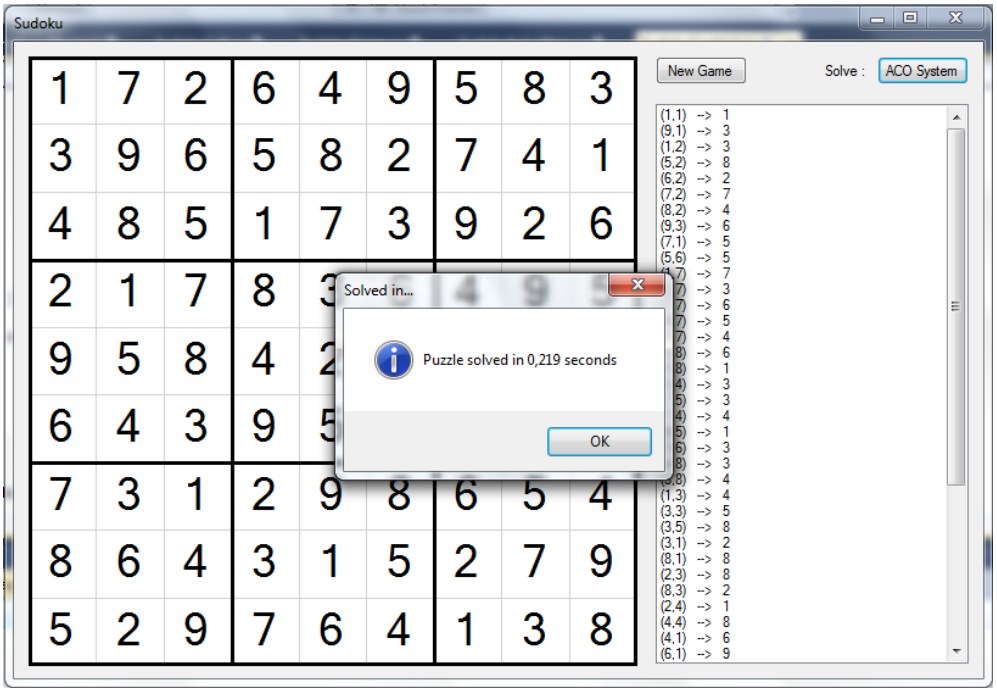
Gambar 4.3 Antarmuka *New Game* (*ACO System*)



Gambar 4.4 Antarmuka *New Game* (Genetika)

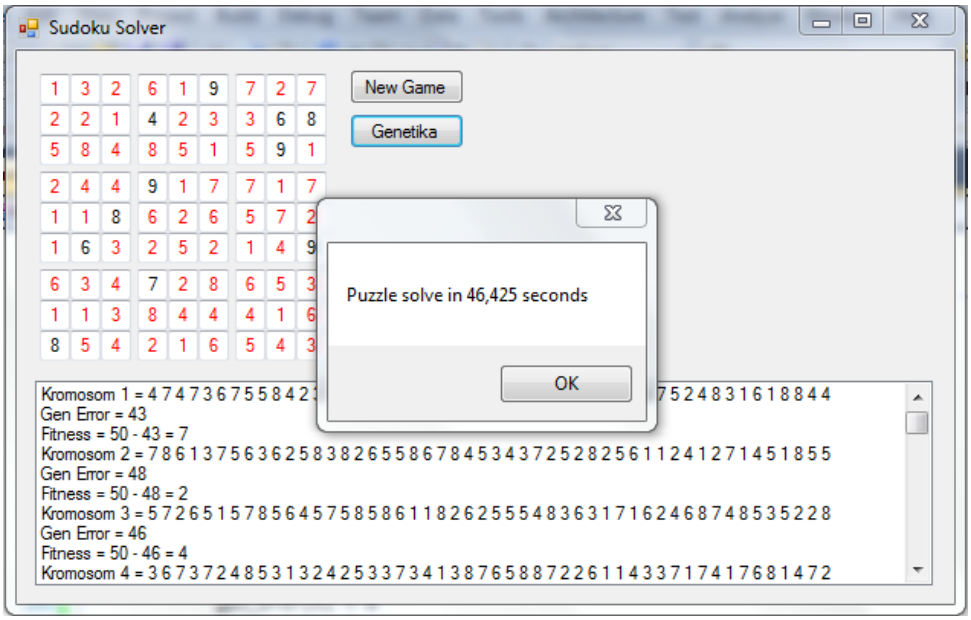
3. Antarmuka *ACO System*

   Antarmuka ini merupakan tampilan setelah pengguna menekan tombol *ACO System*.



Gambar 4.5 Antarmuka *ACO System*

4. Antarmuka Genetika
   Antarmuka ini merupakan tampilan setelah pengguna menekan tombol Genetika.



Gambar 4.6 Antarmuka Genetika

4.2 Pengujian Sistem

Pengujian sistem merupakan hal terpenting yang bertujuan untuk menemukan kesalahan–kesalahan atau kekurangan–kekurangan pada perangkat lunak yang diuji. Pengujian bermaksud untuk mengetahui perangkat lunak yang dibuat sudah memenuhi kriteria yang sesuai dengan tujuan perancangan perangkat lunak tersebut.

4.2.1 Rencana Pengujian

Pengujian dilakukan dengan menguji setiap proses *use case* diagram dan kemungkinan kesalahan yang terjadi untuk setiap proses. Pengujian ini dilakukan dengan teknik pengujian *black box* dan teknik pengujian *white box*.

Tabel 4.4 Rencana Pengujian

No	Komponen Yang Diuji	Detail Pengujian	Jenis Pengujian
1	Algoritma ACO *System*	Proses pencarian langkah optimal dalam penyelesaian permainan Sudoku	*White Box*
2	Algoritma Genetika	Proses pencarian langkah optimal dalam penyelesaian permainan Sudoku	*White Box*
3	*New Game*	Memulai permainan baru dengan *puzzle* Sudoku yang *valid*	*Black Box*
4	*Input Value*	Memasukan angka pada *puzzle* Sudoku	*Black Box*

4.2.2 Pengujian *White Box*

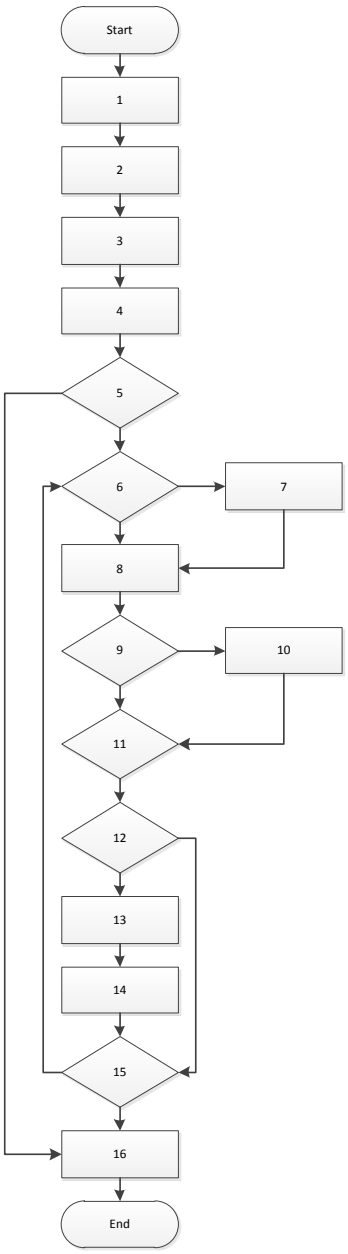
Pengujian *white box* digunakan untuk mengetahui kinerja logika dari algoritma yang telah dianalisis. Ada beberapa tahapan dalam pengujian *white box*, berikut ini merupakan tahapan-tahapannya :

1. Mengubah *pseudocode* menjadi *flowchart*.
2. Mengubah *flowchart* menjadi *flowgraph*.
3. Tahap pengujian, dilakukan dengan 5 cara yaitu menghitung region, menghitung *cyclomatic complexity*, menghitung *independent path*, menggunakan *graph matriks* dan menghitung *predicate node*.

4.2.2.1 Pengujian Algoritma *ACO System*

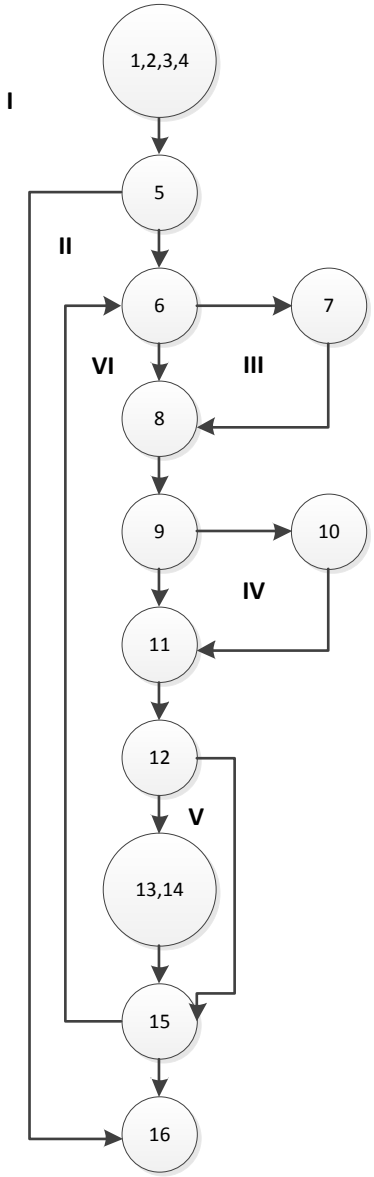
Pseudocode dapat dilihat pada tabel 3.5.

1. **Mengubah *pseudocode* menjadi *flowchart*.**



Gambar 4.7 *Pseudocode* yang diubah menjadi *flowchart*

2. Mengubah *flowchart* menjadi *flowgraph*



Gambar 4.8 *Flowchart* yang diubah menjadi *flowgraph*

3. Tahap Pengujian

a. *Region*

Terdapat 6 *region*

b. *Cyclomatic Complexity*

V(G) = Edge – Node + 2

= 16 -12 + 2

= 4 + 2

= 6

c. *Independent Path*

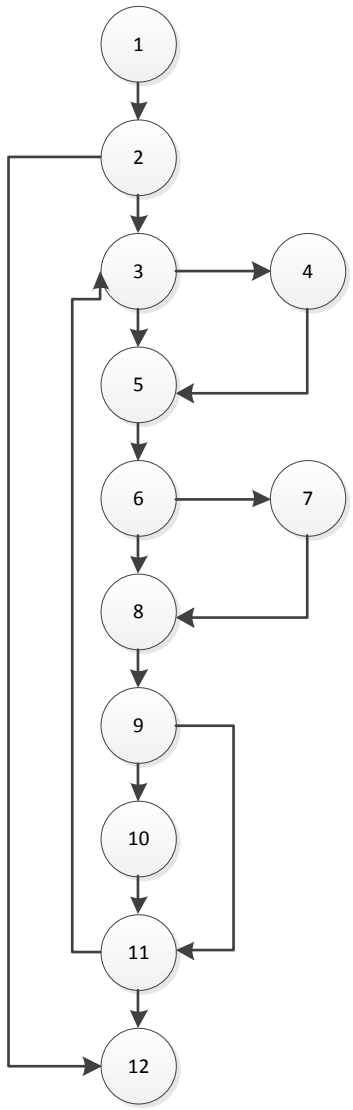
Tabel 4.5 *Independent Path*

No	*Path*
1	1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16
2	1-2-3-4-5-16
3	1-2-3-4-5-6-8-9-11-12-13-14-15-16
4	1-2-3-4-5-6-7-8-9-11-12-13-14-15-16
5	1-2-3-4-5-6-7-8-9-10-11-12-15-16
6	1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-6-7-8-9-10-11-12-13-14-15-16

d. *Graph Matriks*

Tabel 4.6 *Graph Matriks*

Node	1	2	3	4	5	6	7	8	9	10	11	12	
1		1											0
2			1									1	1
3				1	1								1
4					1								0
5						1							0
6							1	1					1
7								1					0
8									1				0
9										1	1		1
10											1		0
11			1									1	1
12													0
Jumlah													5



Gambar 4.9 *Flowgraph* yang diubah menjadi *graph matriks*

e. *Predicate Node*

V(G) = Jumlah *graph matriks* + 1

= 5 + 1

= 6

Tabel 4.7 Pengujian *Path*

No	Kasus Uji	Hasil Yang Diharapkan	Hasil
1	Ketika *predicate node* 2, 3, 6, 9, dan 11 bernilai *true*	Proses akan melanjutkan ke tahap selanjutnya	[✓] Diterima [ ] Ditolak
2	Ketika *predicate node* 2 bernilai *false*	Proses akan langsung menuju ke tahap akhir (dari *node* 2 melanjutkan ke *node* 12)	[✓] Diterima [ ] Ditolak
3	Ketika *predicate node* 3 bernilai *false*	Proses tidak akan melewati ke tahap node 4 (dari *node* 3 melanjutkan ke *node* 5)	[✓] Diterima [ ] Ditolak
4	Ketika *predicate node* 6 bernilai *false*	Proses tidak akan melewati ke tahap node 7 (dari *node* 6 melanjutkan ke *node* 8)	[✓] Diterima [ ] Ditolak
5	Ketika *predicate node* 9 bernilai *false*	Proses tidak akan melewati ke tahap node 10 (dari *node* 9 melanjutkan ke *node* 11)	[✓] Diterima [ ] Ditolak
6	Ketika *predicate node* 11 bernilai *false*	Proses akan melakukan *looping* ke *node* 3	[✓] Diterima [ ] Ditolak

Langkah terakhir dalam melakukan pengujian menggunakan metoda basis *path* testing adalah menyiapkan kasus-kasus uji untuk mengeksekusi semua alur logika yang telah dibuat pada langkah sebelumnya. Kasus uji yang dimaksud disini adalah dengan cara memberikan nilai pada *variable* yang terlibat. Nilai yang dimasukkan haruslah nilai yang mungkin muncul dan sesuai dengan tipe data yang telah didefinisikan. Test case dibuat dalam bentuk tabel yang tujuannya untuk mempermudah eksekusi setiap basis *path*, karena nilai-nilai yang dimasukan dan yang dihasilkan akan lebih mudah diamati.

1. Pengujian *path* 1 = 1-2-3-4-5-6-7-8-9-10-11-12

   Pada *path* ini, supaya *node* 2 dapat menuju ke *node* 3, *node* 3 dapat menuju ke *node* 4, *node* 6 dapat menuju ke *node* 7, *node* 9 dapat menuju ke *node* 10, dan *node* 11 dapat menuju ke *node* 12, maka nilai dari *node* 2, 3, 6, 9, dan 11 harus dapat menghasilkan nilai *true*.

2. Pengujian *path* 2 = 1-2-12

   Pada path ini, supaya *node* 2 dapat menuju ke *node* 12, maka nilai dari *node* 2 harus bernilai *false*.

3. Pengujian *path* 3 = 1-2-3-5-6-7-8-9-10-11-12

   Pada path ini, supaya *node* 3 dapat menuju ke *node* 5, maka nilai dari *node* 3 harus bernilai *false*.

4. Pengujian *path* 4 = 1-2-3-4-5-6-8-9-10-11-12

   Pada path ini, supaya *node* 6 dapat menuju ke *node* 8, maka nilai dari *node* 6 harus bernilai *false*.

5. Pengujian *path* 5 = 1-2-3-4-5-6-7-8-9-11-12

   Pada path ini, supaya *node* 9 dapat menuju ke *node* 11, maka nilai dari *node* 9 harus bernilai *false*.

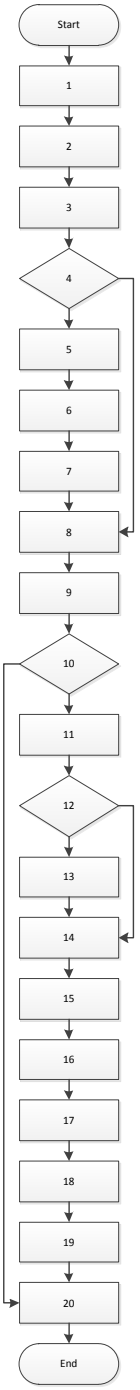
6. Pengujian *path* 6 = 1-2-3-4-5-6-7-8-9-10-11-3-4-5-6-7-8-9-10-11-12

   Pada path ini, supaya *node* 11 dapat menuju ke *node* 3, maka nilai dari *node* 11 harus bernilai *false*.

4.2.2.2 Pengujian Algoritma Genetika

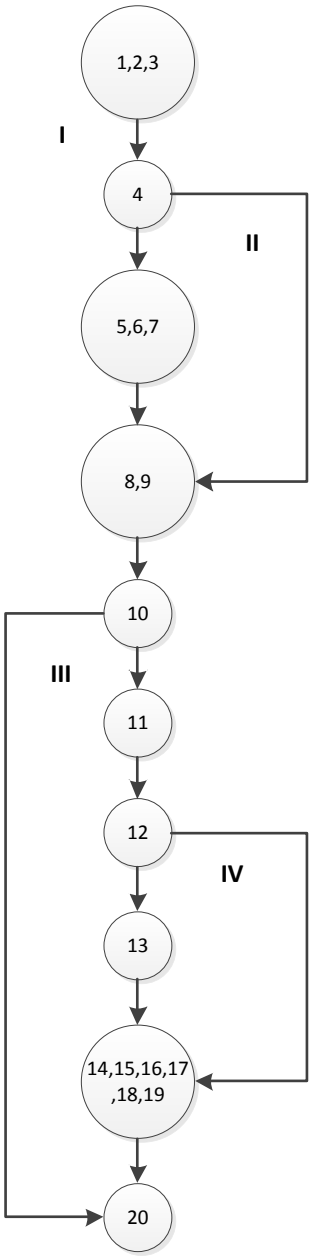
Pseudocode dapat dilihat pada tabel 3.3.

1. **Mengubah *pseudocode* menjadi *flowchart***



Gambar 4.10 *Pseudocode* yang diubah menjadi *flowchart*

2. Mengubah flowchar menjadi flowgraph



Gambar 4.11 *Flowchart* yang diubah menjadi *flowgraph*

3. Tahap Pengujian

a. Region

Terdapat 4 *region*

b. *Cyclomatic Complexity*

V(G) = Edge – Node + 2

= 12 -10 + 2

= 2 + 2

= 4

c. *Independent Path*

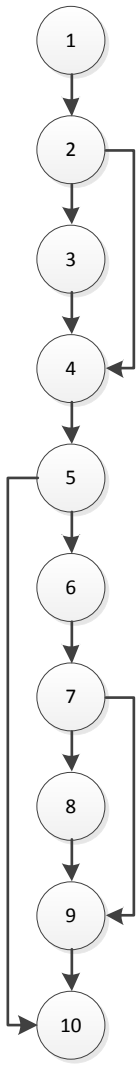
Tabel 4.8 *Independent Path*

No	*Path*
1	1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20
2	1-2-3-4-8-9-10-11-12-13-14-15-16-17-18-19-20
3	1-2-3-4-5-6-7-8-9-10-20
4	1-2-3-4-5-6-7-8-9-10-11-1214-15-16-17-18-19-20

d. *Graph Matriks*

Tabel 4.9 *Graph Matriks*

Node	1	2	3	4	5	6	7	8	9	10	
1		1									0
2			1	1							1
3				1							0
4					1						0
5						1				1	1
6							1				0
7								1	1		1
8									1		0
9										1	0
10											0
Jumlah											3



Gambar 4.12 *Flowgraph* yang diubah menjadi *graph matriks*

e. *Predicate Node*

V(G) = Jumlah *graph matriks* + 1

= 3 + 1

= 4

Tabel 4.10 Pengujian *Path*

No	Kasus Uji	Hasil Yang Diharapkan	Hasil
1	Ketika *predicate node* 2, 5, dan 7 bernilai *true*	Proses akan melanjutkan ke tahap selanjutnya	[✓] Diterima [ ] Ditolak
2	Ketika *predicate node* 2 bernilai *false*	Proses tidak akan melewati ke tahap node 3 (dari *node* 2 melanjutkan ke *node* 4)	[✓] Diterima [ ] Ditolak
3	Ketika *predicate node* 5 bernilai *false*	Proses tidak akan melewati ke tahap node 5 (dari *node* 5 melanjutkan ke *node* 10)	[✓] Diterima [ ] Ditolak
4	Ketika *predicate node* 7 bernilai *false*	Proses tidak akan melewati ke tahap node 8 (dari *node* 7 melanjutkan ke *node* 10)	[✓] Diterima [ ] Ditolak

1. Pengujian *path* 1 = 1-2-3-4-5-6-7-8-9-10

   Pada *path* ini, supaya *node* 2 dapat menuju ke *node* 3, *node* 5 dapat menuju ke *node* 6, dan *node* 7 dapat menuju ke *node* 8, maka nilai dari *node* 2, 5, dan 7 harus dapat menghasilkan nilai *true*.
2. Pengujian *path* 2 = 1-2-4-5-6-7-8-9-10

   Pada path ini, supaya *node* 2 dapat menuju ke *node* 4, maka nilai dari *node* 2 harus bernilai *false*.
3. Pengujian *path* 3 = 1-2-3-4-5-10

   Pada path ini, supaya *node* 5 dapat menuju ke *node* 10, maka nilai dari *node* 5 harus bernilai *false*.
4. Pengujian *path* 4 = 1-2-3-4-5-6-7-9-10

   Pada path ini, supaya *node* 7 dapat menuju ke *node* 9, maka nilai dari *node* 7 harus bernilai *false*.

4.2.3 Pengujian *Black Box*

Pengujian *black box* dilakukan dengan memperhatikan masukan ke sistem dan keluaran dari sistem. Pengujian *black box* yang dilakukan adalah sebagai berikut :

4.2.3.1 Pengujian *New Game*

Tabel 4.11 Pengujian *New Game*

Kasus dan Hasil Uji (Data Normal)			
Data Masukan	**Yang Diharapkan**	**Pengamatan**	**Kesimpulan**
Menekan tombol *New Game*	*Puzzle* baru yang *valid* akan tampil	Menghasilkan *puzzle* baru yang *valid*	[✓] Diterima [ ] Ditolak

4.2.3.2 Pengujian *Input Value*

Tabel 4.12 Pengujian *Input Value*

Kasus dan Hasil Uji (Data Normal)			
Data Masukan	**Yang Diharapkan**	**Pengamatan**	**Kesimpulan**
Angka 0 hingga 9	Jika angka 0 dimasukan maka kotak pada *puzzle* akan kosong, jika angka 1 hingga 9 dimasukan maka akan tampil angka sesuai masukan pada *puzzle*	Angka 0 dimasukan dan menghasilkan kotak kosong pada *puzzle*. Angka 1 hingga 9 dimasukan dan menghasilkan angka sesuai masukan pada *puzzle*	[✓] Diterima [ ] Ditolak
Kasus dan Hasil Uji (Data Salah)			
Data Masukan	**Yang Diharapkan**	**Pengamatan**	**Kesimpulan**
Huruf dan simbol (selain angka 0 hingga 9)	Tidak menampilkan dan merubah apapun pada *puzzle*	*Puzzle* tidak berubah. Tampilan *puzzle* masih seperti semula	[✓] Diterima [ ] Ditolak

4.2.4 Pengujian Performansi

Pengujian yang dilakukan yaitu untuk mengetahui efisiensi waktu dalam pengujian kedua algoritma, adalah sebagai berikut:

Tabel 4.13 Hasil Pengujian Algoritma *ACO System*

Pengujian ke	Waktu (Seconds)
1	0.234
2	0.218
3	0.243
4	0.203
5	0.218
6	0.203

Tabel 4.14 Hasil Pengujian Algoritma Genetika

Pengujian ke	Waktu (Seconds)
1	34.461
2	34.102
3	34.133
4	33.993
5	34.669
6	34.258

4.2.5 Kesimpulan Pengujian

Berdasarkan hasil pengujian, maka dapat ditarik kesimpulan bahwa algoritma *ant colony optimization system* lebih unggul dalam kecepatan dan ketepatan daripada algoritma genetika untuk pencarian langkah optimal dalam penyelesaian permainan Sudoku.