BAB IV
IMPLEMENTASI DAN PENGUJIAN SISTEM

4.1 Implementasi

Implementasi merupakan terjemahan perancangan yang berdasarkan hasil analisis pada bab sebelumnya ke dalam bahasa pemprograman yang dapat dimengerti oleh komputer. Bahasa pemprograman yang digunakan adalah *C#*.

4.1.1 Perangkat Lunak Implementasi dan Pengujian

Perangkat lunak yang digunakan untuk mengimplementasikan dan menguji aplikasi *puzzle* kakuro dengan Algoritma *Harmony Search* adalah sebagai berikut :

Tabel 4. 1 Spesifikasi Perangkat lunak pengujian

Perangkat Lunak	Spesifikasi
Sistem Operasi	Windows 7,
Bahasa pemrograman	Visual Studio 2012

4.1.2 Perangkat Keras Implementasi dan Pengujian

Perangkat keras yang digunakan untuk mengimplementasikan dan menguji aplikasi *puzzle* kakuro dengan Algoritma *Harmony Search* adalah sebagai berikut:

Tabel 4. 2 Spesifikasi Perangkat keras pengujian

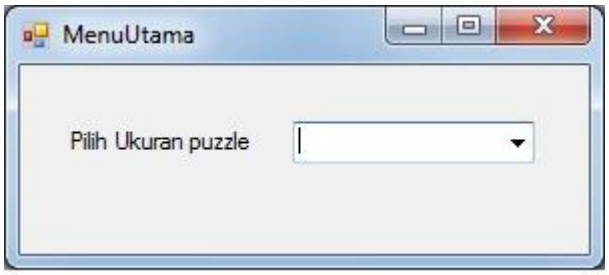
Perangkat Keras	Spesifikasi
Prosesor	Prosesor 3.6GHz QuadCore
VGA	2 GB DDR5
Resolusi Layar	1366 x 768 *Pixel*
Memory	*Memory* 4GB DDR3
Hard Disk	*Hard Disk* 250GB
Keyboard dan *Mouse*	-

4.1.3 Implementasi Antarmuka

Untuk memperjelas bentuk dari implementasi antarmuka berikut tampilan program yang telah dibuat :

1. Tampilan antarmuka *Form* Utama

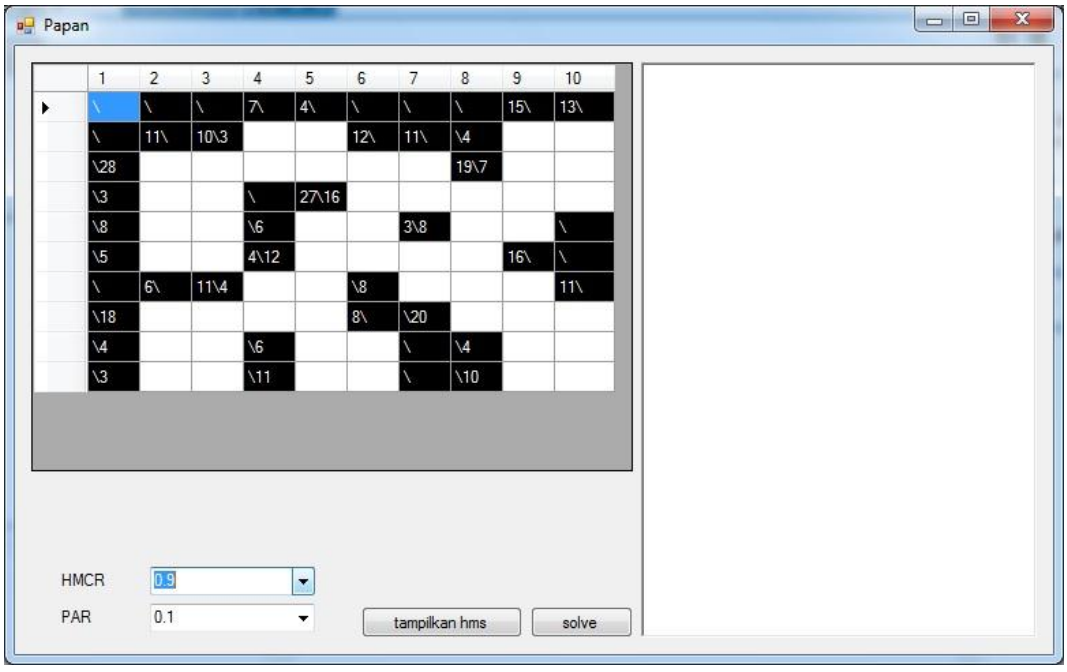
Tampilan antarmuka *Form* Utama memperlihatkan tampilan awal saat aplikasi dijalankan.



Gambar 4.1 Antarmuka *Form* Utama

2. Antarmuka Papan *Puzzle kakuro*

Antarmuka ini merupakan tampilan setelah pengguna memilih ukuran *puzzle*.



Gambar 4.2 Antarmuka Papan *Puzzle* kakuro

4.2 Pengujian Perangkat Lunak

Pengujian perangkat lunak merupakan hal terpenting yang dilakukan untuk menemukan kekurangan atau kesalahan pada perangkat lunak yang diuji. Pengujian bermaksud untuk mengetahui perangkat lunak yang dibuat telah memenuhi kriteria yang sesuai dengan tujuan perancangan perangkat lunak. Adapun metode pengujian yang digunakan pada perangkat lunak ini adalah metode pengujian *black box dan* white box. Pengujian *black box* berfokus pada persyaratan fungsional perangkat lunak, sedangkan *white box* pengujian dengan melihat ke dalam modul untuk meneliti kode-kode program yang ada, dan menganalisis apakah ada kesalahan atau tidak.

4.2.1 Pengujian White Box

Pengujian *white box* merupakan metode perancangan *test case* yang menggunakan struktur control dari perancangan metode dalam mendapatkan *test case.* Adapun metode yang digunakan dalam pengujian *white box* ini adalah metode *basis path.* Metode *basis path* mengijinkan pendesain kasus uji untuk mendapatkan perkiraan lojik yang kompleks dari desain metode dan menggunakan perkiraan ini untuk mendefenisikan aliran eksekusi. Pengujian *white box* akan digunakan pada algoritma *Harmony Search*, untuk mengukur kinerja logika berdasarkan *flowchart* yang telah dibuat pada tahap analisis. Pengujian white box meliputi beberapa tahap yaitu :

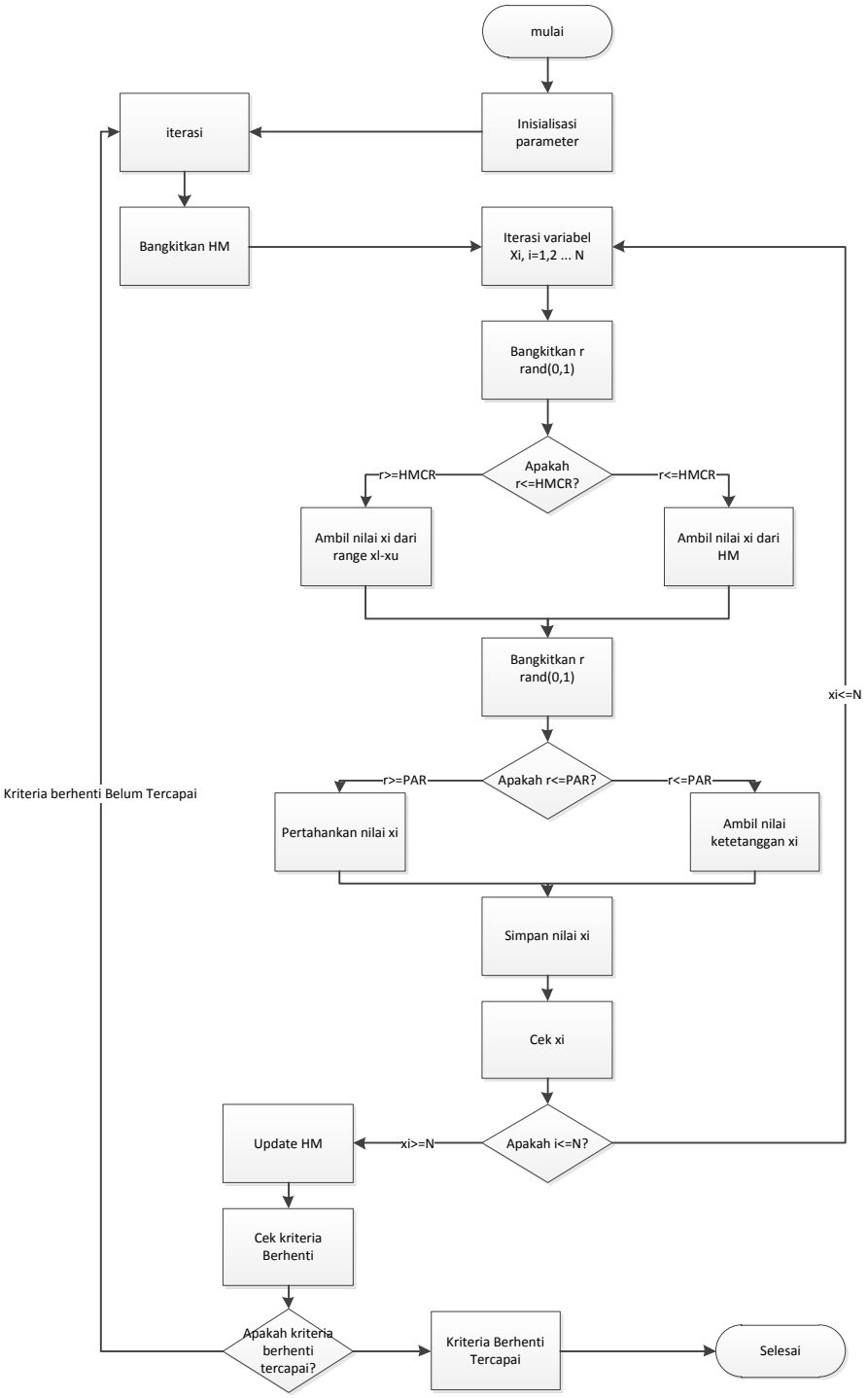
1. Mengubah *pseudocode* menjadi *flowchart*.
2. Mengubah *flowchart* menjadi *flowgraph*.
3. Tahap pengujian, dilakukan dengan 5 cara yaitu menghitung region, menghitung *cyclomatic complexity*, menghitung *independent path*, menggunakan *graph matriks* dan menghitung *predicate node*.

4.2.1.1 Pengujian Algoritma *Harmony Search*

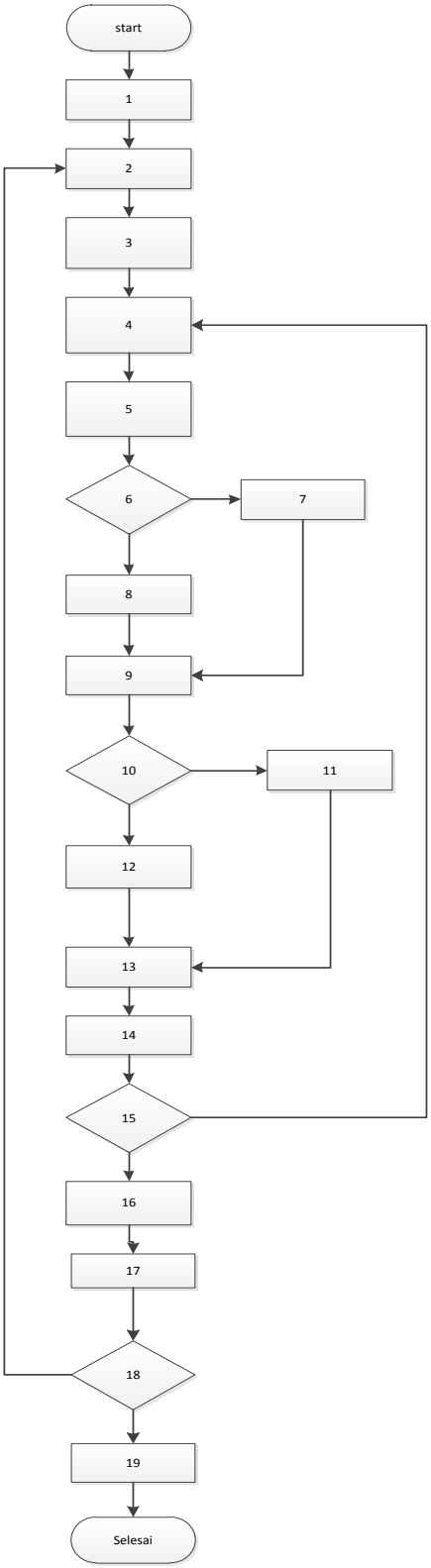
4.2.1.2 Pengujian Algoritma *Harmony Search*

Pseudocode dapat dilihat pada gambar 3.5

1. Mengubah *pseudocode* menjadi *flowchart*

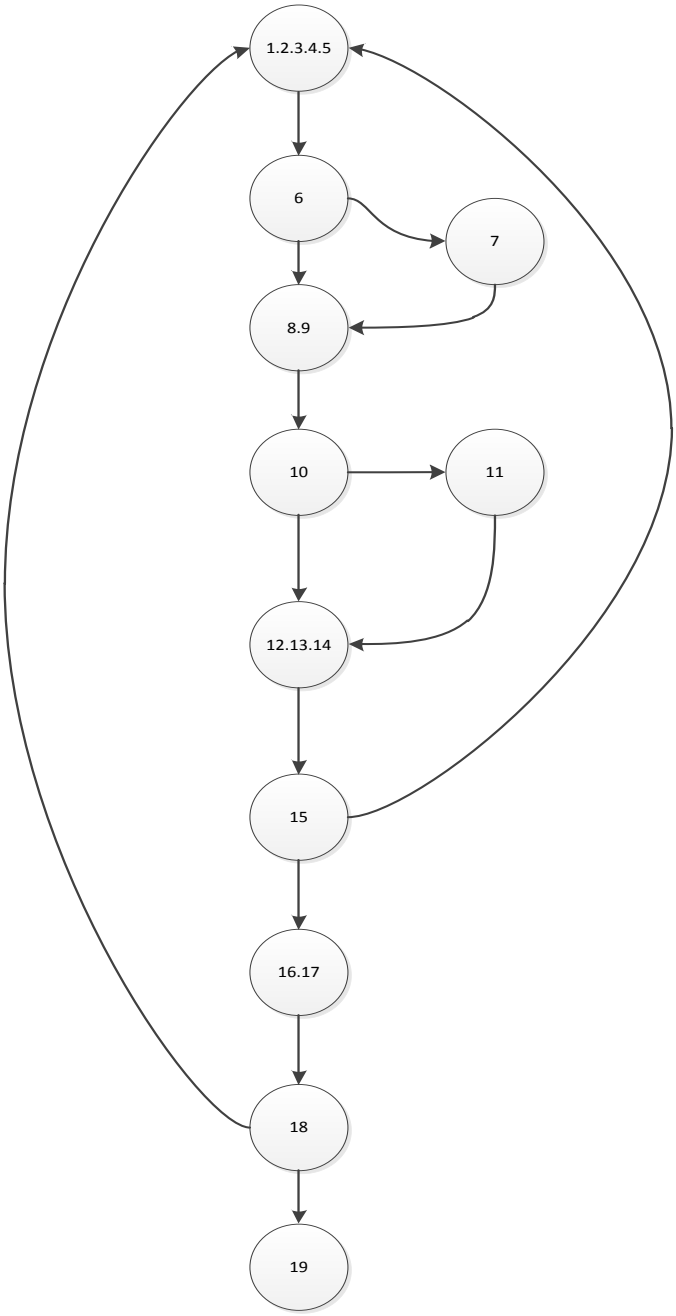


Gambar 4.3 Mengubah *pseudocode* menjadi *flowchart*



Gambar 4.4 penomoran flowchart algoritma *harmony search*

2. Mengubah *flowchart* menjadi *flowgraph*



Gambar 4.5 *flowgraph* algoritma *harmony search*

3. Tahap Pengujian
   a. Region
      Terdapat 5 *region*

b. *Cyclomatic Complexity*

V(G) = Edge – Node + 2

= 14 – 11 + 2

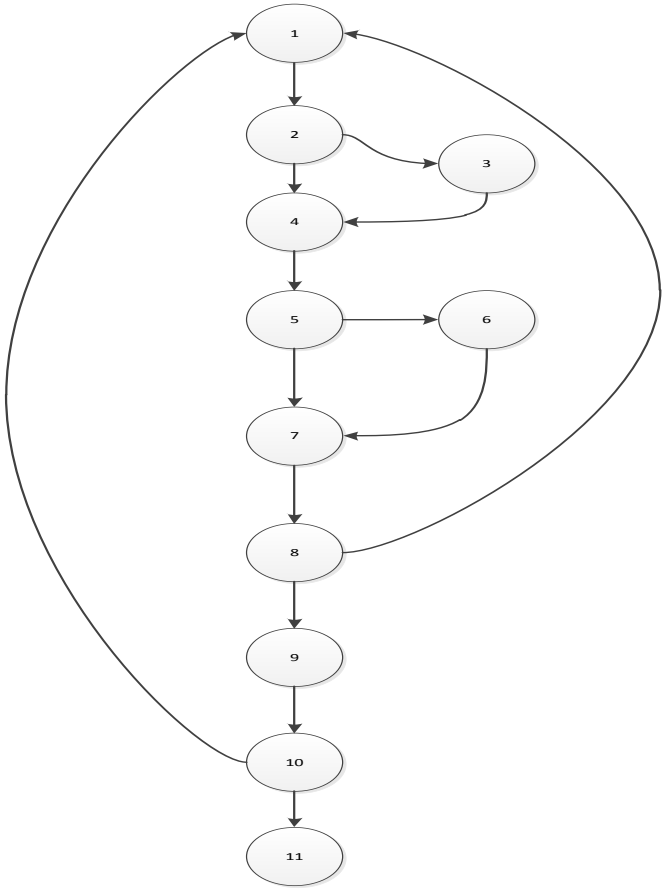
= 5

c. *Independent Path*

Tabel 4. 3 *Independent Path*

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

Penomoran ulang *flowgraph*



Gambar 4.6 Penomoran ulang *flowgraph*

d. *Graph matriks*

Tabel 4. 4 *graph matriks*

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

e. *Predicate Node*

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

= 4 + 1

= 5

Berdasarkan pengujian pada setiap metode, dihasilkan nilai *Cyclomatic Complexity* yang sama yaitu 5. Maka dapat disimpulkan bahwa pengujian *white box* pada algoritma *Harmony Search* berjalan dengan baik, karena setiap pengujian menghasilkan nilai yang sama.

4.2.2 Pengujian Black Box

Pengujian *black box* adalah pengujian yang menyatakan benar bila input yang diberikan akan menghasilkan output yang sesuai dengan spesifikasi program tanpa memperhatikan struktur logika dalam program.

4.2.2.1 Rencana Pengujian

Rencana pengujian yang akan dilakukan pada aplikasi ini dapat dilihat pada Tabel 4.5.

Tabel 4. 5 Rencana Pengujian

Kelas Uji	Butir uji	Jenis Pengujian
MenuUtama	Memilih ukuran *Puzzle*	Black box
Papan	Menjalankan tampil HM	Black box
	Menjalankan algoritma pencarian	Black box
	Menampilkan hasil pencarian	Black box

4.2.2.2 Kasus dan Hasil Pengujian Black Box

Kasus dan hasil pengujian berisi pemaparan dari rencana pengujian yang telah disusun pada scenario pengujian. Pengujian ini dilakukan secara *black box* dengan hanya memperhatikan masukan ke dalam sistem dan keluaran dari masukan tersebut.

1. Pengujian MenuUtama

Pengujian ini dilakukan untuk menguji persyaratan fungsional *Form* MenuUtama yaitu memilih ukuran *puzzle*. Kasus dan hasil uji dengan data normal dapat dilihat pada Tabel 4.6.

Tabel 4. 6 Pengujian MenuUtama

Kasus dan Hasil Uji (Data Normal)			
Data Masukan	**Yang Diharapkan**	**Pengamatan**	**Kesimpulan**
memilih ukuran *puzzle*	Menampilkan papan *puzzle* sesuai pilihan	Menampilkan papan *puzzle* sesuai pilihan	[√] diterima [ ] ditolak

2. Pengujian Papan

Pengujian ini dilakukan untuk menguji persyaratan fungsional *Form* Papan yang terdiri dari menjalankan tampil HM, menjalankan algoritma pencarian, dan menampilkan hasil pencarian. Kasus dan hasil uji dengan data normal dapat dilihat pada Tabel 4.7.

Tabel 4. 7 Pengujian Papan

Kasus dan Hasil Uji (Data Normal)			
Data Masukan	**Yang Diharapkan**	**Pengamatan**	**Kesimpulan**
menjalankan tampil HM	Menampilkan *harmony memory* pada setiap kotak	Tombol dapat berfungsi sesuai yang diharapkan	[√] diterima [ ] ditolak
menjalankan algoritma pencarian	Menjalankan algoritma untuk mencari solusi dan menampilkan progressnya ke logbox	Tombol dapat berfungsi sesuai yang diharapkan	[√] diterima [ ] ditolak
menampilkan hasil pencarian	Menampilkan hasil pencarian pada papan *puzzle*	Dapat menampilkan hasil pencarian pada papan *puzzle*	[√] diterima [ ] ditolak

4.2.3 Pengujian Parameter

Parameter aplikasi yang akan diuji yaitu waktu pencarian dan perbandingan HMCR dan PAR.

Pengujian dilakukan menggunakan beberapa konfigurasi seperti pada tabel 4.4.

Tabel 4. 8 konfigurasi pengujian parameter

Grid *Puzzle*	HMCR	PAR
6x6	0,7	0,1
		0,2
		0,3
	0,8	0,1
		0,2
		0,3
	0,9	0,1
		0,2
		0,3

Grid Puzzle	HMCR	PAR
7x7	0,7	0,1
		0,2
		0,3
	0,8	0,1
		0,2
		0,3
	0,9	0,1
		0,2
		0,3
8x8	0,7	0,1
		0,2
		0,3
	0,8	0,1
		0,2
		0,3
	0,9	0,1
		0,2
		0,3
9x9	0,7	0,1
		0,2
		0,3
	0,8	0,1
		0,2
		0,3
	0,9	0,1
		0,2
		0,3
10x10	0,7	0,1
		0,2
		0,3
	0,8	0,1
		0,2
		0,3
	0,9	0,1
		0,2
		0,3

Pengujian dilakukan dengan jumlah iterasi maksimal 1000. Hasil pengujian waktu dihitung dalam satuan detik.

Tabel 4. 9 hasil pengujian Algoritma *harmony search*

Grid *Puzzle*	HMCR	PAR	Waktu Solusi		
			Percobaan ke-1	Percobaan ke-2	Percobaan ke-3
6x6	0,7	0,1	0.066	0.052	0.063
		0,2	0.071	0.060	0.068
		0,3	0.092	0.096	0.089
	0,8	0,1	0.059	0.062	0.053
		0,2	0.098	0.064	0.071
		0,3	0.133	0.099	0.092
	0,9	0,1	0.042	0.053	0.043
		0,2	0.053	0.086	0.062
		0,3	1.112	0.078	0.060
7x7	0,7	0,1	0.058	0.060	0.062
		0,2	0.064	0.062	0.067
		0,3	Solusi Tidak Ditemukan	Solusi Tidak Ditemukan	1.632
	0,8	0,1	0.050	0.060	0.049
		0,2	Solusi Tidak Ditemukan	0.065	0.057
		0,3	0.067	0.088	0.079
	0,9	0,1	0,037	0.036	0.040
		0,2	0.043	0.042	0.054
		0,3	0.056	Solusi Tidak Ditemukan	0.084
8x8	0,7	0,1	0,082	0.073	Solusi Tidak Ditemukan
		0,2	Solusi Tidak Ditemukan	0,09	0.165
		0,3	0.096	0.092	Solusi Tidak Ditemukan
	0,8	0,1	0.063	0.057	0.062
		0,2	0.075	0.068	0.095
		0,3	Solusi Tidak Ditemukan	Solusi Tidak Ditemukan	Solusi Tidak Ditemukan
	0,9	0,1	0.055	0.060	0.050
		0,2	0.068	0.070	0,09
		0,3	Solusi Tidak Ditemukan	2.325	Solusi Tidak Ditemukan

Grid *Puzzle*	HMCR	PAR	Waktu Solusi		
			Percobaan ke-1	Percobaan ke-2	Percobaan ke-3
9x9	0,7	0,1	0.113	0.158	0.158
		0,2	Solusi Tidak Ditemukan	0.230	0.099
		0,3	0.145	Solusi Tidak Ditemukan	Solusi Tidak Ditemukan
	0,8	0,1	0.085	0.0116	0.094
		0,2	0.218	0.195	0.091
		0,3	0.088	Solusi Tidak Ditemukan	3.564
	0,9	0,1	0.052	0.054	0,042
		0,2	0.058	0.065	0.067
		0,3	0.087	0.072	Solusi Tidak Ditemukan
10x10	0,7	0,1	2.155	Solusi Tidak Ditemukan	Solusi Tidak Ditemukan
		0,2	1.862	Solusi Tidak Ditemukan	2.112
		0,3	Solusi Tidak Ditemukan	Solusi Tidak Ditemukan	Solusi Tidak Ditemukan
	0,8	0,1	1.219	0.962	0.938
		0,2	1.136	0.975	0.994
		0,3	Solusi Tidak Ditemukan	2.864	Solusi Tidak Ditemukan
	0,9	0,1	0.096	0.165	0.235
		0,2	0.462	0.853	1.134
		0,3	Solusi Tidak Ditemukan	Solusi Tidak Ditemukan	1.267

Dari hasil uji coba di atas memperlihatkan nilai yang berbeda – beda setiap kali algoritma dijalankan. Permasalahan ini disebabkan hampir seluruh proses dalam algoritma *harmony search* mengandalkan teknik acak.

Dalam proses implementasi dilakukan penambahan fungsi untuk mengecek nilai setiap kotak jawaban yang diilih. Itu dilakukan karena algoritma *harmony search* hanya mengecek nilai berdasarkan hasil dari perhitungan fungsi objektif per vektor solusi. Maka banyak kemungkinan terdapat nilai duplikat. Berdasarkan hal tersebut maka disisipkan suatu fungsi untuk mengecek nilai yang dibangkitkan oleh algoritma *harmony search*, fungsi ini mengecek kelegalan suatu nilai dalam setiap kotak, sehingga lebih meminimalisir nilai dulikat pada setiap

lajurnya (baik baris mauun kolom). Ketika setiap nilai pada lajur telah legal, maka barulah nilai – nilai itu dihitung berdasarkan lajurnya menggunakan fungsi objektif.

Dalam beberapa percobaan, algoritma tidak berhasil menemukan solusi atau terjebak dalam *local optimum*, hal ini terjadi terutama bila jumlah kotak jawaban banyak dan nilai HMCR terlalu kecil, karena dengan nilai HMCR terlalu kecil probabilitaas kemungkinan nilai r *random* lebih kecil dari HMCR lebih kecil sehingga pemilihan *xnew* diambil dari batas atas sampai batas bawah bukan dari HM. Begitu juga ketika nilai PAR terlalu besar maka probabilitas kemungkinan nilai r random lebih kecil dari PAR akan semakin besar, sehingga nilai *xnew* akan digantikan oleh nilai tetangga. Jika terjebak *local optimum*, proses akan berjalan terus menerus dan kecil kemungkinan untuk menemukan solusi, karena nilai *xnew* bukan berasal dari HM. Berdasarkan data pada tabel 4.9 didapat bahwa dengan HMCR 0.9 atau HMCR besar masih bisa menghasilkan solusi dan dengan PAR 0.1 atau PAR kecil.

4.2.4 Kesimpulan Pengujian

Dari hasil pengujian dapat disimpulkan pula bahwa Algoritma *harmony search* berhasil menemukan solusi dengan waktu yang cepat. Jumlah kotak jawaban sangat berpengaruh pada solusi yang dihasilkan dan waktu pencarian. Semakin banyak jumlah kotak jawaban maka semakin sulit untuk menghasilkan solusi dan waktu pencarian akan semakin lama. Dalam beberapa percobaan, algoritma tidak berhasil menemukan solusi atau terjebak dalam *local optimum*, sehingga proses akan berjalan terus menerus dan kecil kemungkinan untuk menemukan solusi. Sehingga pada kasus terburuk, performansi Algoritma *harmony search* tidak lebih baik bila dibandingkan dengan algoritma lain, karena terdapat kemungkinan terjebak dalam *local optimum*.