Chapter 5

A Genetic Algorithm for Graph Matching using Graph Node Characteristics\(^1\)\(^2\)

Graph Matching has attracted the exploration of applying new computing paradigms because of the large number of applications it has spawned. Soft computing and evolutionary computation have tremendous potential for offering efficient solutions to the graph matching problem. Soft computing techniques mimic the approach of human beings and different techniques such as neural networks, fuzzy logic, support vector machines etc are available. These techniques are very much useful in intelligent applications viz: the applications that need intelligence. Many applications in recognition, vision, speech and text understanding make use of these techniques and are found to be very successful. Soft computing techniques have been also employed along with graphs in various graph theoretic applications including, graph matching [Jain, 2005].

Evolutionary computation techniques such as genetic algorithms, genetic programming, ant colony optimization etc, have found applications in many problem domains. Evolutionary computational techniques are fundamentally search techniques which search for efficient optimal solutions in the state space. By posing the problems in different domains as search problems the evolutionary computational techniques can be applied to almost any problem area. This chapter presents a novel application of evolutionary computational paradigm namely the genetic algorithm for exact graph matching/ graph isomorphism.

\(^1\) Parts of this chapter appear in, “A Simple Genetic Algorithm for Graph Matching using Graph Node Characteristics”, Communicated.
\(^2\) Parts of this chapter appear in, “A Steady State Genetic Algorithm for Graph Matching using Graph Vertex Properties”, Communicated.
5.1 Introduction

Evolutionary computation is a new computational paradigm that has emerged as a subfield of computational intelligence and is mainly employed for solving combinatorial optimization problems. The evolutionary computational techniques include [W11];

- Evolutionary algorithms such as genetic algorithms, genetic programming etc.
- Techniques based on swarm intelligence such as ant colony optimization, bees’ algorithm, cuckoo search etc.
- The other algorithms such as Teaching-Learning based algorithms, simulated annealing and many others.

Of the various evolutionary computational techniques, the evolutionary algorithms have become very popular and have been employed for solving various problems. Amongst the evolutionary algorithms, the genetic algorithm technique has been very general and is being used for solving many an optimization problem. Genetic Algorithm was first introduced by John Holland in 1960’s at University of Michigan [Mitchell, 1998] and its power was published to the world in 1975, through the seminal work of Holland, “Adaptation in Natural and Artificial Systems”, [Holland, 1992]. The paradigm of genetic algorithms was made popular by David Goldberg who presented its various applications in his book, “Genetic Algorithms in Search, Optimization, and Machine Learning” in 1989 [Goldberg, 1989]. The genetic algorithms have been used in searching optimum solutions in problems as diverse as bioinformatics, phylogenetics, computer science, economics, chemistry, manufacturing, mathematics, physics and other fields.

Informally one can describe genetic algorithms as a programming technique that imitates biological evolution for solving problems. Given a problem to solve; the genetic algorithm finds an optimal solution by using a set of randomly generated solutions and evolving better solutions in iterations using operators which mimic the natural evolution process. The
operators that imitate the natural evolution process are derived from the theory of evolution proposed by Charles Darwin (Survival of the fittest) [W15].

Genetic algorithms also have been used in conjunction with graph theory to solve many problems. Genetic algorithms and graphs have been used for solving machine scheduling problem [Nirmala G., 2012]. Genetic algorithm has been employed in distribution system planning and is very often used in finding matching in bipartite graphs.

Different genetic algorithm solutions to the inexact graph matching variants have been discussed in [Ravadanegh, 2008]. In this chapter a novel genetic algorithm based technique is proposed for solving the exact graph matching (graph isomorphism) problem using the graph node properties.

The methodology makes use of the vertex invariance, degree invariance and summation of shortest distance invariance (as described in chapter 2) for ascertaining the exact graph matching. Once the graphs are proved to be isomorphic, a novel genetic algorithm is employed for finding the vertex correspondence between the two isomorphic graphs. A new chromosome structure using the real numbers for describing the various characteristics of the nodes is devised. The chromosome describes the proposed vertex correspondences between the two graphs. A new fitness function that computes the fitness of each chromosome (to be precise the optimality of vertex correspondence) is devised. This is based on corollary 5.1 described in section 5.2. Further the crossover and mutation operators for the proposed chromosome representation are defined. Using the above representations and the operators defined, the optimal vertex correspondence is found using both simple/ generational GA and the steady state GA.

The methodology is tested on a large pairs of synthetic graphs and the results are very encouraging as the accurate results are obtained in almost all of the cases.

The chapter is organized into five sections. The section 5.2 gives the complete description of the methodology devised to find the matching between two undirected unweighted graphs.
The section 5.3 describes the genetic algorithm and its various components. The experimentation conducted using the genetic algorithms developed and the analysis is presented in section 5.4. Section 5.5 gives the summary.

5.2 The Overall Methodology

The graph matching (graph isomorphism) problem is approached in two stages in the proposed methodology. First the two graphs are processed for finding whether they are isomorphic. The isomorphism is verified by employing the invariance in terms of the

- **The number of vertices**
- **The degree invariance**, i.e. the vertices of same degree are equal in number in both the graphs.
- **The sum of shortest distance invariance**. The number and value of sum of shortest distances from a vertex to other vertices are the same in the two graphs.

Using the above invariants the two graphs are verified for isomorphism. If the two graphs are not similar or isomorphic the methodology concludes that the two graphs are not similar. But if the two graphs are similar then newly devised genetic algorithm is employed for establishing the correspondence between the vertices of the two graphs. The genetic algorithm is the second stage of the methodology.

The two graphs are represented as adjacency matrices; the degree and shortest distance sum from each vertex to all other vertices are computed and used for proving the similarity of graphs. Further the eccentricity of each node is computed and used along with the vertex degree and sum of shortest distances to other vertices for generating an initial population of chromosomes, which represents the initial estimates to the vertex correspondence between the two graphs. A newly defined fitness function estimates the fitness of each individual in terms of correspondence of vertices. This fitness function is based on corollary 5.1.
Corollary 5.1

Given two isomorphic graphs $G_1$ and $G_2$, if there is a relation $R$ between vertices of $G_1$ to vertices of $G_2$ such that $R: G_1 \rightarrow G_2$ or $R: G_1(v_i) \rightarrow G_2(v_j)$ then if

$$Val = \frac{1}{\sum(\Delta_i\sigma_i e_i - \Delta_i\sigma_i e_i) + k} = \frac{1}{k}$$

Then the vertices $i$ of $G_1$ corresponds to vertices $j$ of $G_2$.

Proof: Now when $Val$ assumes this value then $\sum(\Delta_i\sigma_i e_i - \Delta_i\sigma_i e_i) = 0$ and as in theorem 4.1 the vertices $i$ of $G_1$ correspondences to vertices $j$ of $G_2$. Hence, the proof of the corollary.

Once the initial population is generated then the next generation of the population or the next iteration in the genetic algorithm is generated using the simple GA (generational GA) or steady state GA technique. The newly defined selection, crossover and mutation operators are employed for generating the offspring’s (new solutions) to the vertex correspondence problem. After iterating through the genetic algorithm, the fitness function is employed to select the most-fit individual as the proposed correspondence between vertices of the graphs. The complete methodology is depicted pictorially in the flowchart given in Figure 5.1. The process is also brought out in the two stage algorithm given below.

Algorithm 5.1: Graph Matching GA ($G_1$, $G_2$)

Stage I
1: Read the edges of $G_1$ and construct $X(G_1)$, similarly read edges of $G_2$ and construct $X(G_2)$.

2. Find the degrees of vertices of both graphs, say $D_1$ and $D_2$

3. Find sum of shortest distances from a vertex to all the other vertices in both the graphs say $Sp_1$, $Sp_2$. 

98
4. Find the eccentricity of vertices of both graphs say $E_1$ and $E_2$.

5. Check for invariance between $(D_1, D_2)$, and $(Sp_1, Sp_2)$ respectively. If invariance does not hold, display the two graphs are not isomorphic and stop. Otherwise display the two graphs are isomorphic and go to step 6. (Stage 2)

**Stage 2**

6. Construct initial population of chromosomes representing various initial solutions of vertex correspondence.

7. Find the fitness of fittest individual after the termination of the iterations using simple/generational GA and steady state GA separately

8. Display the vertex correspondence, using the fittest individual obtained from step 7.

9. Stop

The genetic algorithm components and the operations are described in detail in section 5.3.
Stage 1

Start

Read the adjacency matrices of graph $G_1$ & $G_2$ (say $X_1$ & $X_2$) $n_1$ & $n_2$ be the no. of vertices of graphs $G_1$ & $G_2$

Find the degree of each of the vertices of graph $G_1$ & $G_2$ ($D_1$ & $D_2$)

Find the sum of shortest distances to other vertices in the graphs. $Sp_1$ & $Sp_2$

Find the eccentricity of every vertex of each graph. $E_1$ & $E_2$

$n_1 \neq n_2$

Yes

Print the graphs are not isomorphic

Stop

No

Degree invariance

Yes

Print the graphs are isomorphic

No

Sum of shortest distance invariance

Yes

No

A

Figure 5.1: Flowchart (Contd...)
5.3 The Genetic Algorithm for Vertex Correspondence

The genetic algorithm for correspondence match between vertices of the two graphs requires various components for its functioning viz;

- A chromosome representation, that gives a proposed solution to the problem.
- The crossover and mutation operators that are reproduction operators and are employed for producing off springs during the functioning of the GA.
- The selection operator helps in selection of parents for reproduction.
• The **fitness function** that evaluates the fitness of individual solutions and generally is the objective function that is to be optimized by the candidate solutions.

The genetic algorithm literature presents two basic types of genetic algorithms, namely the simple or generational GA and the second is the Steady State GA [Goldberg, 1989]. The steady state GA replaces the least fit individual from the population by the newly generated chromosomes if their fitness is better than the least fit individual’s fitness in the population, whereas the simple GA generates a complete new generation of the population that replaces the older one. The different components that constitute the genetic algorithm for graph matching are described in the following sub sections.

**5.3.1 The Chromosome Representation**

Generally the candidate solution for any problem that is solved using genetic algorithms is represented as a chromosome. The chromosome consists of many parts of the solution which are referred in literature as genes. Further the names of the individual values in the genes are called as alleles. Many different representation methodologies for chromosome are available. The more popular ones are the binary string representation or the real value representation [Goldberg, 1989].

For the problem of graph matching a new chromosome representation that employs the real values of the degree of the node, the summation of shortest distance to all the vertices from a given vertex and the eccentricity of the vertex is employed. The chromosome contains n genes, where n is the number of vertices/ nodes of the graph. Every gene contains eight alleles that specify the possible correspondences and their characteristic values. The first four alleles stand for the vertex number, degree and summation of shortest distance to every other vertex and the eccentricity of the vertex from graph $G_1$. The second set of four alleles stands for the same information from the second graph $G_2$ which is corresponding to the vertex in the first graph. The pictorial representation of the
A typical representation of an example graph pair is given in Figure 5.3 and Figure 5.4.

**Legend**

- $V_i^j$: Vertex number $i$ from $G_1$
- $V_j^i$: Vertex number $j$ from $G_2$
- $D_i^j$: Degree of vertex $V_i$ in graph $G_1$
- $D_j^i$: Degree of vertex $V_j$ in graph $G_2$
- $E_i^j$: Eccentricity of $V_i$ in graph $G_1$
- $E_j^i$: Eccentricity of $V_j$ in graph $G_2$
- $Sp_i^j$: Sum of shortest path distance to all the other vertices from $V_i$ in graph $G_1$
- $Sp_j^i$: Sum of shortest path to all the other vertices from $V_j$ in graph $G_2$

**Figure 5.2**: The Chromosome Representation

<table>
<thead>
<tr>
<th>Vertices</th>
<th>Deg</th>
<th>SP</th>
<th>Ecc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>

**Figure 5.3**: Graph $G_1$ and Parameters
A typical chromosome for graphs $G_1$ (Figure 5.3) and graph $G_2$ (Figure 5.4) is given in Figure 5.5.

\[
\{13524262\}{23523352\}{42621262\}{52625212\}{32622352\}
\]

**Figure 5.5**: Chromosome Representation for Graphs $G_1$ and $G_2$

### 5.3.2 The Selection Operator

The literature has references to many selection operators such as elitism, round robin, greedy, stochastic and roulette wheel selection [Goldberg, 1989]. Each of the selection operators has advantages and also disadvantages. The roulette wheel selection operator which is used in the genetic algorithms for graph matching in this work is described in the following.

The roulette wheel selection operator selects the individual from a pool of individuals based on the randomized proportionality. Every individual of the population is evaluated for the proportionality / weightage. Each individual is assigned a weightage based on the ratio of its frequency to the total number of occurrences. During selection a uniform random number is generated and is used to find its occurrence among the cumulative probability values, which is used for selecting a parent. This functioning of the operator is depicted in Figure 5.6. In the roulette wheel, shown in the Figure 5.6 the random number generated is between 0 to 9. As shown in the figure if the random number generated is 0 or 1, individual
1 is selected, if the random number generated is between 2 to 5 individual 2 is selected and so on.

![Diagram of Roulette Wheel Selection](image)

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Population size = 5
Fitness of 1 = 0.2
2 = 0.4
3 = 0.2
4 = 0.1
5 = 0.1

**Figure 5.6: Roulette Wheel Selection**

**Note:** The individuals are assigned boxes on the wheel based on their weightage and individuals with higher weightage have more chance of being selected. In this work the weightage to each chromosome is assigned based on its fitness value.

The simple GA also uses the elitism selection along with roulette wheel selection; it involves selecting the fittest individual from the population.

### 5.3.3 The Crossover Operator

The crossover operator is the reproduction operator involving two parents. The two parents are selected using the selection operator and are subjected to crossover operation based on a crossover probability. In this work the crossover probability is assumed to be 0.9. The crossover operator is defined as in Figure 5.7.

The crossover operation defined is a single point crossover. The single point crossover involves randomly selecting crossover points in the two parents and then exchanging the two genes. If the gene so exchanged has alleles already present in the other genes of the
chromosome, the alleles of such genes are replaced with alleles in the outgoing gene of this parent. This is clearly depicted in Figure 5.7.

| Parent 1 | \[V_1^{11}D_1^{11}Sp_1^{11}E_1^{11}V_2^{11}D_2^{11}Sp_2^{11}E_2^{11}\] (\[V_1^{31}D_1^{31}Sp_1^{31}E_1^{31}V_2^{31}D_2^{31}Sp_2^{31}E_2^{31}\]) |
| Parent 2 | \[V_1^{41}D_1^{41}Sp_1^{41}E_1^{41}V_2^{41}D_2^{41}Sp_2^{41}E_2^{41}\] (\[V_1^{51}D_1^{51}Sp_1^{51}E_1^{51}V_2^{51}D_2^{51}Sp_2^{51}E_2^{51}\]) |
| Crossover parameters | \[CP1 = 3\] \[CP2 = 4\] also \[V_1^{11} = V_1^{42}\] \[V_1^{22} = V_1^{51}\] |
| Child 1 | \[V_1^{31}D_1^{31}Sp_1^{31}E_1^{31}V_2^{31}D_2^{31}Sp_2^{31}E_2^{31}\] (\[V_1^{21}D_1^{21}Sp_1^{21}E_1^{21}V_2^{21}D_2^{21}Sp_2^{21}E_2^{21}\]) |
| Child 2 | \[V_1^{31}D_1^{31}Sp_1^{31}E_1^{31}V_2^{31}D_2^{31}Sp_2^{31}E_2^{31}\] (\[V_1^{52}D_1^{52}Sp_1^{52}E_1^{52}V_2^{52}D_2^{52}Sp_2^{52}E_2^{52}\]) |

**Note:** The bold allele indicates the changed values

*Figure 5.7: The Single Point Crossover operator*

### 5.3.4 The Mutation Operator

The mutation operator is introduced to add variety to the population by randomly varying the genes of the chromosome. The mutation operator is applied with lesser mutation probability. The mutation operator involves randomly selecting the gene for mutation. Once the gene for mutation is selected the first vertex (first allele) is modified by randomly selecting the vertex and finding the corresponding values for the next three alleles. If the newly introduced vertex is present in any other gene of the chromosome then it is replaced.
by the alleles of the mutated vertex. The complete mutation operator is depicted in Figure 5.8.

<table>
<thead>
<tr>
<th>Offspring</th>
<th>( (V_1^1, D_1^1, S_1^1, E_1^1), V_2^1, D_2^1, S_2^1, E_2^1) (V_1^2, D_1^2, S_1^2, E_1^2), V_2^2, D_2^2, S_2^2, E_2^2) (V_1^3, D_1^3, S_1^3, E_1^3), V_2^3, D_2^3, S_2^3, E_2^3) (V_1^4, D_1^4, S_1^4, E_1^4), V_2^4, D_2^4, S_2^4, E_2^4) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mutation parent</td>
<td>Mutation position : 3 old vertex repl = ( V_1^3 )</td>
</tr>
<tr>
<td></td>
<td>Mutation Location : 1 new vertex = ( V ) corresponding ( D, S, E ) and other parameters</td>
</tr>
<tr>
<td></td>
<td>Let ( V = V_1^2 ) then replace ( V_1^2 ) and corresponding parameters with ( V_1^3 ) and corresponding parameters</td>
</tr>
<tr>
<td>Mutated Offspring</td>
<td>( (V_1^1, D_1^1, S_1^1, E_1^1), V_2^1, D_2^1, S_2^1, E_2^1) (V_1^2, D_1^2, S_1^2, E_1^2), V_2^2, D_2^2, S_2^2, E_2^2) (V_1^3, D_1^3, S_1^3, E_1^3), V_2^3, D_2^3, S_2^3, E_2^3) (V_1^4, D_1^4, S_1^4, E_1^4), V_2^4, D_2^4, S_2^4, E_2^4) )</td>
</tr>
</tbody>
</table>

**Note:** The bold allele indicates the changed values.

**Figure 5.8:** The Mutation Operator

### 5.3.5 The Fitness Function

The fitness function is a very important component of the genetic algorithm. It decides on the fitness of the individual chromosomes which are the probable solutions to the problem. The fitness function is also used as an important condition for terminating the GA. In most of the cases the fitness function is the objective function of combinatorial optimization. In the present problem of graph matching the fitness of the proposed solution/ chromosome is obtained by the summation of difference between the products of the three parameters of the corresponding vertices in the chromosome. More particularly the difference between the products of the degree, summation of shortest distances to all other vertices in the graph and the eccentricity of the corresponding vertices in a gene is considered and a small value delta is added to this difference (empirically taken as 0.01). The reciprocal of this term is the fitness value of the chromosome. The fitness function is depicted in equation 5.1.
\[ f = \frac{1}{\sum_{i=1}^{N} \left( |D_{v1g1}^{i}SP_{v1g1}^{i}ECC_{v1g1}^{i} - D_{v2g2}^{i}SP_{v2g2}^{i}ECC_{v2g2}^{i}| + \delta \right)} \quad \ldots(5.1) \]

Where \( \delta \) is 0.01

This fitness function returns a value of 100 when the chromosome gives the required solution. This is because when the chromosome gives the required solution then the products of the matching vertices are the same and hence their sum is zero. The fitness function is devised based on the corollary 5.1 described in section 5.2.

The two different genetic algorithms employed for vertex correspondence matching are described in ensuing sections.

5.3.6 The Simple or Generational GA

This technique of the genetic algorithm is the simplest and involves generation of the complete population in a single iteration. The simple GA is historically found to be a very efficient search technique and has been employed for finding solutions to various problems. The algorithm employed for simple or generational genetic algorithm to search for vertex correspondence is given as algorithm 5.2 in the following.

Algorithm 5.2: Simple or Generational Genetic Algorithm

Input: The adjacency matrices of the graphs \( G_1 \) and \( G_2 \), the degrees of all the vertices of \( G_1 \) and \( G_2 \) (\( D_1, D_2 \)), the summation of shortest distances from a vertex to all other vertices in \( G_1 \) and \( G_2 \) (\( Sp_1, Sp_2 \)), the eccentricities of all the vertices in \( G_1 \) and \( G_2 \) (\( E_1, E_2 \))

Output: The correspondence of vertices in the graphs \( G_1 \) and \( G_2 \).

Step 1: Start

Step 2: Find the quantity of the initial population. \( \text{Init} = \text{Fact}(n)/5 \) if \( n \leq 5 \), \( \text{Init} = \text{Fact}(n)/20 \) if \( n > 5 \) and \( \leq 7 \), \( \text{Init} = 250 \) otherwise.

Step 3: Generate the initial random population of size equal to \( \text{Init} \), let it be CR
Step 4: Find the fitness of all the individuals

Step 5: \( \text{iter}=0; \)

Step 6: While \( \text{iter} \leq 15000 \) do

   Step a: Find the individual with maximum fitness. Let the maximum fitness be \( M \)

   Step b: If \( M \) is greater than or equal to predefined optimal fitness of 100, break out of the loop.

Step c: \( K=0 \)

Step d: While \( (K<\text{Init}) \) do

   Step i: Copy the fittest individual to the new generation being created, CRN by incrementing the index \( K \). If \( K>\text{Init} \) Break out of loop.

   Step ii: Select a parent using roulette wheel selection, say \( P_1 \)

   Step iii: Select a parent using roulette wheel selection, say \( P_2 \)

   Step iv: Perform cross over operation and generate two offsprings, say \( CH_1, CH_2 \) using crossover probability

   Step v: Perform Mutation operation on the offsprings \( CH_1 \) and \( CH_2 \) using mutation probability

   Step vi: ADD \( CH_1 \) to the new population CRN by incrementing \( K \). If \( K > \text{Init} \) Break out of the loop

   Step vii: ADD \( CH_2 \) to the new population CRN by incrementing \( K \). If \( K > \text{Init} \) Break out of the loop

Step e: Copy CRN to CR

Step f: Find fitness of all the individuals of CR

Step 7: Select the chromosome with maximum fitness, and display the correspondence of the vertices between graphs \( G_1 \) and \( G_2 \).

Step 8: Stop.
The simple GA has been implemented and tested on a large number of synthetic graph pairs and the results of vertex correspondences are satisfactory.

### 5.3.7 The Steady State GA

The steady state GA progresses slowly from one generation of population to the next. It makes use of the selective replacement of the least fit individuals from the population. The steady state GA is more suitable when there are more number of constraints. The complete steady state genetic algorithm is depicted as Algorithm 5.3.

**Algorithm 5.3: Steady State Genetic Algorithm**

**Input:** The adjacency matrices of the graphs $G_1$ and $G_2$, the degrees of all the vertices of $G_1$ and $G_2$ ($D_1, D_2$), the summation of shortest distances from a vertex to all other vertices in $G_1$ and $G_2$ ($Sp_1, Sp_2$), the eccentricities of all the vertices in $G_1$ and $G_2$ ($E_1, E_2$)

**Output:** The correspondence of vertices in the graphs $G_1$ and $G_2$.

**Step 1:** Start

**Step 2:** Find the quantity of the initial population. $Init=\text{Fact} (n)/5$ if $n<=5$, $Init = \text{Fact}(n)/20$ if $n>5$ and $<=7$, $Init=250$ otherwise.

**Step 3:** Generate the initial random population of size equal to $Init$, let it be $CR$

**Step 4:** Find the fitness of all the individuals

**Step 5:** $\text{iter}=0$

**Step 6:** While $\text{iter} <= 150000$ do

a. Select a parent using roulette wheel selection, say $P_1$

b. Select a parent using roulette wheel selection, say $P_2$

c. Perform cross over operation and generate two offsprings, say $CH_1, CH_2$ using crossover probability

d. Perform Mutation operation on the offsprings $CH_1$ and $CH_2$ using mutation probability
e. Replace the least fit individual of the population by CH1 and CH2, only if the fitness of the offsprings CH1 and CH2 are greater than the least fit individuals of the population.
f. Find the individual with maximum fitness. Let the maximum fitness be M.
g. If M is greater than predefined optimal fitness of 100, break out of the loop.

**Step 7:** Select the chromosome with maximum fitness, and display the correspondence of the vertices between graphs G₁ and G₂.

**Step 8:** Stop.

The steady state GA has been implemented using MATLAB and tested on a large variety of synthetic graphs and the results have been satisfactory. Comparisons of the results of the two types of genetic algorithm are provided in section 5.4.

### 5.4 Experimentation

The proposed method of graph matching using the invariants such as degree of vertex, shortest distance sum from a vertex to other vertices and vertex eccentricity for checking graph similarity and further new genetic algorithm for finding node correspondence has been implemented using MATLAB. The methodology has been thoroughly tested using a large number of synthetic graphs. Table 5.1 lists the different parameters of the genetic algorithm (GA) used in this implementation. The results of the methodologies including the time taken for about thirty four graph pairs are listed in Table 5.2.

**Table 5.1:** The parameters of the Genetic Algorithms

<table>
<thead>
<tr>
<th>Sl No.</th>
<th>Particulars</th>
<th>Simple GA</th>
<th>Steady State GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Crossover Probability</td>
<td>0.9</td>
<td>0.9</td>
</tr>
<tr>
<td>2</td>
<td>Mutation Probability</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>3</td>
<td>Maximum Iteration</td>
<td>150000</td>
<td>150000</td>
</tr>
<tr>
<td>4</td>
<td>Initial Population</td>
<td>250</td>
<td>250</td>
</tr>
<tr>
<td>5</td>
<td>Maximum Fitness</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>
Table 5.2: The Experimentation Data of the Graph Matching Using Genetic Algorithms

<table>
<thead>
<tr>
<th>Sl No</th>
<th>No. of Vertices of the Graph</th>
<th>No. of Edges of Graph G1</th>
<th>No. of Edges of Graph G2</th>
<th>ISO (Y/ N)</th>
<th>Corr. Mapping (Y/ N)</th>
<th>Simple GA</th>
<th>Steady State GA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>No. of Iter.</td>
<td>Time</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>y</td>
<td>y</td>
<td>1</td>
<td>0.08</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>y</td>
<td>y</td>
<td>1</td>
<td>0.172</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>y</td>
<td>y</td>
<td>3</td>
<td>0.36</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>y</td>
<td>y</td>
<td>1</td>
<td>0.20</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>y</td>
<td>y</td>
<td>1</td>
<td>0.34</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>y</td>
<td>y</td>
<td>1</td>
<td>0.33</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>y</td>
<td>y</td>
<td>1</td>
<td>0.28</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>y</td>
<td>y</td>
<td>5</td>
<td>0.94</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>y</td>
<td>y</td>
<td>3</td>
<td>2.47</td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>y</td>
<td>y</td>
<td>3</td>
<td>3.66</td>
</tr>
<tr>
<td>11</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>y</td>
<td>y</td>
<td>1</td>
<td>0.563</td>
</tr>
<tr>
<td>12</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>y</td>
<td>y</td>
<td>2</td>
<td>2.30</td>
</tr>
<tr>
<td>13</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>y</td>
<td>y</td>
<td>2</td>
<td>1.78</td>
</tr>
<tr>
<td>14</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>y</td>
<td>y</td>
<td>4</td>
<td>4.47</td>
</tr>
<tr>
<td>15</td>
<td>8</td>
<td>12</td>
<td>12</td>
<td>y</td>
<td>y</td>
<td>1</td>
<td>0.64</td>
</tr>
<tr>
<td>16</td>
<td>9</td>
<td>11</td>
<td>11</td>
<td>y</td>
<td>y</td>
<td>4</td>
<td>4.03</td>
</tr>
<tr>
<td>17</td>
<td>9</td>
<td>11</td>
<td>11</td>
<td>y</td>
<td>y</td>
<td>8</td>
<td>8.91</td>
</tr>
<tr>
<td>18</td>
<td>9</td>
<td>11</td>
<td>11</td>
<td>y</td>
<td>y</td>
<td>8</td>
<td>10.47</td>
</tr>
<tr>
<td>19</td>
<td>9</td>
<td>11</td>
<td>11</td>
<td>y</td>
<td>y</td>
<td>9</td>
<td>10.25</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
<td>12</td>
<td>12</td>
<td>y</td>
<td>y</td>
<td>7</td>
<td>6.65</td>
</tr>
<tr>
<td>21</td>
<td>10</td>
<td>12</td>
<td>12</td>
<td>y</td>
<td>y</td>
<td>12</td>
<td>11.25</td>
</tr>
<tr>
<td>22</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td>y</td>
<td>y</td>
<td>14</td>
<td>12.14</td>
</tr>
<tr>
<td>23</td>
<td>11</td>
<td>13</td>
<td>13</td>
<td>y</td>
<td>y</td>
<td>11</td>
<td>10.344</td>
</tr>
<tr>
<td>24</td>
<td>11</td>
<td>13</td>
<td>13</td>
<td>y</td>
<td>y</td>
<td>14</td>
<td>13.875</td>
</tr>
<tr>
<td>25</td>
<td>11</td>
<td>13</td>
<td>13</td>
<td>y</td>
<td>y</td>
<td>16</td>
<td>23.97</td>
</tr>
<tr>
<td>26</td>
<td>14</td>
<td>13</td>
<td>13</td>
<td>y</td>
<td>y</td>
<td>24</td>
<td>41.33</td>
</tr>
<tr>
<td>27</td>
<td>15</td>
<td>16</td>
<td>16</td>
<td>y</td>
<td>y</td>
<td>21</td>
<td>29.43</td>
</tr>
<tr>
<td>28</td>
<td>20</td>
<td>16</td>
<td>16</td>
<td>y</td>
<td>y</td>
<td>1</td>
<td>295.63</td>
</tr>
<tr>
<td>29</td>
<td>20</td>
<td>28</td>
<td>28</td>
<td>y</td>
<td>y</td>
<td>51</td>
<td>107.1</td>
</tr>
<tr>
<td>30</td>
<td>22</td>
<td>24</td>
<td>24</td>
<td>Y</td>
<td>Y</td>
<td>61</td>
<td>131.59</td>
</tr>
<tr>
<td>31</td>
<td>24</td>
<td>27</td>
<td>27</td>
<td>Y</td>
<td>Y</td>
<td>61</td>
<td>145.30</td>
</tr>
<tr>
<td>32</td>
<td>26</td>
<td>32</td>
<td>32</td>
<td>Y</td>
<td>Y</td>
<td>88</td>
<td>221.84</td>
</tr>
<tr>
<td>33</td>
<td>28</td>
<td>36</td>
<td>36</td>
<td>Y</td>
<td>Y</td>
<td>79</td>
<td>222.14</td>
</tr>
<tr>
<td>34</td>
<td>30</td>
<td>40</td>
<td>40</td>
<td>Y</td>
<td>Y</td>
<td>144</td>
<td>326.67</td>
</tr>
</tbody>
</table>
The plot of number of iterations v/s the number of nodes and the time taken for finding the solution v/s the number of nodes is given in Figures 5.9 and 5.10 respectively.

**Figure 5.9:** The Plot of Number of Iterations V/s Nodes for Simple and Steady State GA

**Figure 5.10:** The Plot of Time for Finding the Solution V/s Nodes for Simple and Steady State GA
The Genetic algorithm based approaches use a technique similar to methodology described in Chapter 4 for verifying isomorphism, but employ new GA approaches to find vertex correspondence. The time complexity of the GA depends on population size and the number of generations. The detailed time complexity analysis is given in Chapter 6. Among the two methodologies the simple GA is found to be better than steady state GA in terms of time taken by the algorithm to find approximate solution as depicted in the Figure 5.10.

The overall results of graph matching (for isomorphic graph pairs) using the two different genetic algorithms are listed in Table 5.3 and plotted in Figure 5.11. The figure shows both the methodologies have performed well in terms of the correctness.

**Table 5.3: The Overall Results**

<table>
<thead>
<tr>
<th>Sl No</th>
<th>Methodology</th>
<th>No. of Graph Pairs Tested</th>
<th>Simple GA</th>
<th>Steady State GA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>No. of Graphs Correctly identified</td>
<td>Percentage Efficiency</td>
</tr>
<tr>
<td>1</td>
<td>Graph Matching using Simple GA</td>
<td>29</td>
<td>29</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>Graph Matching using Steady State GA</td>
<td>29</td>
<td>29</td>
<td>100</td>
</tr>
</tbody>
</table>

![Figure 5.11: The Plot of Overall Efficiency of Genetic Algorithms](image-url)
5.5 Summary

In this chapter a novel method of graph matching as a two stage process is proposed. In the first stage the graphs are checked for similarity and if they are similar, vertex correspondences are obtained using the newly devised genetic algorithm. Genetic algorithm has been previously employed for bipartite matching applications but not for exact graph matching (graph isomorphism) of simple undirected graphs. This chapter has introduced a novel method for the purpose. The methodology has made use of the characteristics of the vertices of the graph and a new fitness function that finds the optimal vertex correspondences by computing the difference of the products of the characteristics namely degrees, shortest distance sum and eccentricity.

The methodology proposed in this chapter is found to be robust and has performed very well on all the graph pairs. Some of the typical graphs conventionally employed in graph isomorphism testing (counter examples) have been used and the results are as expected. The genetic algorithm proposed here is a novel one and employs a new chromosome representation and a novel fitness function. The results are very encouraging and can be further employed for various applications.