# Chapter 12

# Solving Graph Matching with EDAs Using a Permutation-Based Representation

E. Bengoetxea
*Department of Computer Architecture and Technology*
*University of the Basque Country*
endika@si.ehu.es

P. Larrañaga
*Department of Computer Science and Artificial Intelligence*
*University of the Basque Country*
ccplamup@si.ehu.es

I. Bloch A. Perchant
*Department of Signal and Image Processing*
*Ecole Nationale Supérieure des Télécommunications*
{bloch, perchant}@tsi.enst.fr

**Abstract** Graph matching has become an important area of research because of the potential advantages of using graphs for solving recognition problems. An example of its use is in image recognition problems, where structures to be recognized are represented by nodes in a graph that are matched against a model, which is also represented as a graph.

As the number of image recognition areas that make use of graphs is increasing, new techniques are being introduced in the literature. Graph matching can also be regarded as a combinatorial optimization problem with constraints and can be solved with evolutionary computation techniques such as Estimation of Distribution Algorithms.

This chapter introduces for the first time the use of Estimation of Distribution Algorithms with individuals represented as permutations to solve a particular graph matching problem. This is illustrated with the real problem of recognizing human brain images.

P. Larrañaga et al. (eds.), *Estimation of Distribution Algorithms*

**Keywords:** Inexact Graph Matching, Estimation of Distribution Algorithms, Human Brain Images

# 1. Introduction

Representation of structural information by graphs is widely used in domains that include network modelling, psycho-sociology, image interpretation, and pattern recognition. There, graph matching is used to identify nodes and therefore structures. Most existing problems and methods in the graph matching domain assume graph isomorphism, where both graphs being matched have the same number of nodes and links. For some problems, this bijective condition between the two graphs is too strong and it is necessary to weaken it and express the correspondence as an inexact graph matching problem.

Examples of inexact graph matching can be found in the pattern recognition field, where structural recognition of images is performed: the model (also called the atlas or map depending on the application) is represented in the form of a graph, where each node contains information for a particular structure, and data graph are generated from the images to be analyzed. Graph matching techniques are then used to determine which structure in the model corresponds to each of the structures in a given image. When the data graph is generated automatically from the image to be analyzed, the difficulty of accurately segmenting the image into meaningful entities means that oversegmentation techniques need to be applied (Perchant et al., 1999; Perchant and Bloch, 1999; Perchant, 2000). These ensure that the boundaries between the meaningful entities to be recognized will appear in the data image as clearly distinct structures. As a result, the number of nodes in the data graph increases and isomorphism condition between the model and data graphs cannot be assumed. Such problems call for inexact graph matching, and similar examples can be found in other fields. There, the graph matching technique of choice has to perform the recognition process by returning a solution where each node in the data graph is matched with the corresponding node in the model graph.

In addition, another important aspect to be taken into account is the fact that some graph matching problems contain additional constraints on the matching that have to be satisfied in order to consider the matching as correct.

The complexity of the graph matching problem is mostly determined by the size of the model and data graphs. This has been proved to be NP-hard (Lovàsz and Plummer, 1986), and therefore the use of heuristic methods is justified.

Different techniques have been applied to inexact graph matching, including combinatorial optimization (Cross and Hancock, 1999; Cross et al., 1997; Singh and Chaudhury, 1997), relaxation (Finch et al., 1997; Gold and Rangarajan, 1996; Hancock and Kittler, 1990; Wilson and Hancock, 1996; Wilson and Hancock, 1997), the EM algorithm (Cross and Hancock, 1998; Finch et al.,

1998), and Evolutionary Computation techniques such as Genetic Algorithms (GAs) (Boeres et al., 1999; Myers and Hancock, 2001).

This chapter proposes optimization through learning and simulation of probabilistic graphical models (such as Bayesian networks and Gaussian networks) as the method of choice. Adaptations of different Estimation of Distribution Algorithms (EDAs) for use in inexact graph matching are also introduced. EDAs are also modified to deal with additional constraints in a graph matching problem. Existing articles on using EDAs to solve the graph matching problem are Bengoetxea et al. (2000a) and Bengoetxea et al. (2000b), which compare EDAs with GAs in their use for this type of problem.

The outline of this chapter is as follows: Section 2 explains the graph matching problem, showing it as a combinatorial optimization problem with constraints. Section 3 proposes a permutation-based approach for solving the inexact graph matching problem using EDAs. Sections 4 and 5 introduce a method for translating from individuals containing a permutation to valid solutions of the inexact graph matching problem for both discrete and continuous domains. Section 6 describes the experiment carried out and the results obtained. Finally, Section 7 gives conclusions and suggests further work.

## 2. Graph matching as a combinatorial optimization problem with constraints

In any combinatorial optimization problem an important influence on algorithm performance is the way that the problem is defined, in both the representation of individuals chosen, and the fitness function used to evaluate those individuals. This section gives some examples of representations (the encoding of points in the search space).

### 2.1 Representation of individuals

One of the most important tasks in defining any problem to be solved with heuristics is choosing an adequate representation of individuals, because this determines to a large extent the performance of the algorithms. An individual represents a solution, i.e. a point in the search space that has to be evaluated. For a graph matching problem, each individual represents a match between the nodes of a data graph $G_2$ and those of model graph $G_1$.

A representation of individuals for this problem that was used in GAs in Boeres et al. (1999) that could also be applied to EDAs is the following: individuals with $|V_1| \cdot |V_2|$ binary (only contains 0s and 1s) genes or variables, where $V_1$ and $V_2$ are the number of nodes in graphs $G_1$ and $G_2$ respectively. In each individual, the meaning of entry $c_{ij}$, $1 \leq i \leq |V_1|$ and $1 \leq j \leq |V_2|$, is the following: $c_{ij} = 1$ means that the $j^{th}$ node of $G_2$ is matched with the $i^{th}$ node of $G_1$. The main drawback of this type of representation is the large number of

variables or genes that the individual contains, which increases the complexity of the problem that EDAs or GAs have to solve. The cardinality of the search space is also

$$2^{|V_1|\cdot|V_2|}, \tag{12.1}$$

which is quite large, although not all the individuals are valid (there are some restrictions to consider within the individuals).

Another possible representation that can be used either in GAs or EDAs consists of individuals which each contains $|V_2|$ genes or variables, where each variable can contain any value between 1 and $|V_1|$. More formally, the individual as well as the solution it represents could be defined as follows: for $1 \leq k \leq |V_1|$ and $1 \leq i \leq |V_2|$, $X_i = k$ means that the $i^{th}$ node of $G_2$ is matched with the $k^{th}$ node of $G_1$. This is the representation used for instance in Bengoetxea et al. (2000a) and Bengoetxea et al. (2000b). In this representation, the number of possible solutions to the inexact graph matching problem is given by the following formula for number of cases of permutation with repetition:

$$\sum_{i_1=1}^{|V_2|-|V_1|-1} \cdots \sum_{i_{|V_1|}=1}^{|V_2|-|V_1|-1} \frac{|V_2|!}{\prod_{k=1}^{|V_1|} i_k!} \tag{12.2}$$

where the values $i_k$ ($k = 1, \ldots, |V_1|$) satisfy the condition $\sum_{k=1}^{|V_1|} i_k = |V_2|$. We will refer to this representation later in Section 6 as *traditional.*

An example of the *traditional* representation of individuals is shown in Figure 12.1 for a particular example where the model graph $G_1$ contains 6 nodes (labeled from 1 to 6) and the data graph $G_2$ represents a segmented image and contains 11 nodes (labeled from 1 to 11). This individual represents a solution (a point in the search space) where the first two nodes of $G_2$ are matched to node number 1 of $G_1$, the next four nodes of $G_2$ are matched to node number 2 of $G_1$, and so on.

| 1 | 1 | 2 | 2 | 2 | 2 | 3 | 4 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|---|---|---|

*Figure 12.1* Traditional representation of an individual for the problem of graph matching, when $G_1$ (the model graph) contains 6 nodes and $G_2$ (the data graph representing the segmented image) contains 11 nodes.

Another important aspect that determines which individual representation is the most appropriate is given by the fact that every problem has restrictions that have to be satisfied by the solutions (i.e. the individuals) in order to be considered as *correct* or useful. For instance, when applying graph matching techniques for the recognition of human brain structures, it is important for any acceptable solution that all the main brain structures such as the cerebellum are identified (e.g. a solution where the cerebellum is not present in the brain

image could not be accepted!). Each particular problem has its own particular constraints, and the different representations of individuals chosen have to take these into account. The reader can find a review of types of individual representations as well as the resolution of the restrictions in the human brain problem in Bengoetxea et al. (2000a). The same reference introduces different methods and mechanisms for generating correct individuals that satisfy these constraints. It is important to note that for each different individual representation the procedure to handle those constraints is different, and therefore this aspect has to be taken into account in any representation in order to obtain correct solutions and to minimize the complexity of the problem.

# 3. Representing a matching as a permutation

Individual representations based on permutations have been typically applied to problems such as the Traveling Salesman Problem or the Vehicle Routing Problem, where either a salesman or a vehicle has to pass through a number of places at the minimum cost.

A *permutation-based representation* can also be used for problems such as inexact graph matching. In this case the meaning of the individual is completely different, as an individual does not show directly which node of $G_2$ is matched with each node of $G_1$. In fact, what we obtain from each individual is the order in which nodes will be analyzed and treated so as to compute the matching that it is representing.

For the individuals to contain a permutation, the individuals will be the same size as the *traditional* ones described in Section 2.1 (i.e. $|V_2|$ variables long). However, the number of values that each variable can obtain will be of size $|V_2|$, and not $|V_1|$ as in that representation. In fact, it is important to note that a permutation is a list of numbers in which all the values from 1 to $n$ have to appear in an individual of size $n$. In other words, our new representation of individuals need to satisfy a strong constraint in order to be considered as correct, that is, they all have to contain every value from 1 to $n$, where $n = |V_2|$.

More formally, all the individuals used for our problem of inexact graph matching will be formed from $|V_2|$ genes or variables, that contain no repeated value within the individual and have values between 1 and $|V_2|$. For $1 \leq k \leq |V_2|$ and $1 \leq i \leq |V_2|$, $X_i = k$ means that the $k^{th}$ node of $G_2$ will be the $i^{th}$ node that is analyzed for its most appropriate match.

## 3.1 From the permutation to the solution it represents

Once the type of individuals have been formally defined, we need to create a method to obtain a solution from the permutation itself because the representation does not directly define the meaning of the solution. Every individual

requires this step in order to be evaluated. As a result, it is important that this translation is performed by a fast and simple algorithm.

This section introduces a way of performing this step. A solution for the inexact graph matching problem can be calculated by comparing the nodes to each other and deciding which is more similar to which using a similarity function $\varpi(i,j)$ defined for this purpose to compute the similarity between nodes $i$ and $j$. The similarity measures used so far in the literature have been applied to two nodes, one from each graph, and their goal has been to help in the computation of the fitness of a solution, that is, the final value of a fitness function. However, the similarity measure $\varpi(i,j)$ proposed in this section is quite different, as these two nodes to be evaluated are both in the data graph $(i,j \in V_2)$. With these new similarity values we will be able to look for the node in $G_2$ which is most similar to any particular node that is also in $G_2$. The aim of this is to identify for each particular node of $G_2$ which other nodes in the data graph are most similar to it, and try to group it with the best set of already matched nodes.

We have not defined the exact basis for the similarity measure $\varpi$ yet. Different aspects could be taken into account, and this topic will be further discussed in Section 3.3.

As explained in the introduction, each particular problem usually contains specific constraints that have to be satisfied by all the proposed solutions. If this is the case, another important aspect is to ensure that the solution represented by a permutation is always a correct individual. A solution will be considered as correct only when it satisfies the conditions defined for the problem. In order to set restrictions on our problem and test how the optimization methods handle them, we will assume in this chapter that the only condition to consider an individual as correct is that all the nodes of $G_2$ have to be matched with a node of $G_1$, and that every node of $G_1$ is matched with at least one node of $G_2$. These conditions will be satisfied by the translation procedure proposed next for both discrete and continuous domains.

Given an individual $\boldsymbol{x}= (x_1, \ldots, x_{|V_1|}, x_{|V_1|+1}, \ldots, x_{|V_2|})$, the procedure to do the translation is performed in two phases as follows:

- The first $|V_1|$ values $(x_1, \ldots, x_{|V_1|})$ that directly represent nodes of $V_2$ will be respectively matched to nodes 1, 2..., $|V_1|$ (that is, the node $x_1 \in V_2$ is matched with the node $1 \in V_1$, the node $x_2 \in V_2$ is matched with the node $2 \in V_1$, and so on, until the node $x_{|V_1|} \in V_2$ is matched with the node $|V_1| \in V_1$).

- For each of the following values of the individual, $(x_{|V_1|+1}, \ldots, x_{|V_2|})$, and following their order of appearance in the individual, the most similar node will be chosen from all the previous values in the individual by means of the similarity measure $\varpi$. For each of these nodes of $G_2$, we

**From discrete permutations to the solution**

**Definitions**

$|V_1|$: number of nodes in the model graph $G_1$
$|V_2|$: number of nodes in the data graph $G_2$
$|V_2| > |V_1|$.
$n = |V_2|$: size of the individual (the permutation)
$\boldsymbol{x} = (x_1, \ldots, x_{|V_2|})$: individual containing a permutation
$x_i \in \{1, \ldots, n\}$: value of the $i^{th}$ variable in the individual
$PV_i = \{x_1, \ldots, x_{i-1}\}$: set of values assigned in the individual to the variables $X_1, \ldots, X_{i-1}$ ($PV$ = previous values)
$\varpi(i, j)$: similarity function that measures the similarity of node $i$ with respect to node $j$

**Procedure**

*Phase 1*
For $i = 1, 2, \ldots, |V_1|$
(first $|V_1|$ values in the individual, treated in order)
Match node $x_i \in V_2$ of data graph $G_2$
with node $i \in V_1$ in model graph $G_1$

*Phase 2*
For $i = |V_1| + 1, \ldots, |V_2|$
(remaining values in the individual, treated in this order)
Let $k \in PV_i$ be the most similar node to $x_i$ from
all the nodes of $PV_i$ ($k = \max_{j=1 \ldots i-1} \varpi(i, j)$)
Match node $x_i \in V_2$ of data graph $G_2$
with the matched node that is matched to node $k$ of $G_2$

*Figure 12.2* Pseudocode to compute the solution represented by a permutation-based individual.

assign the matched node of $G_1$ that is matched to the most similar node of $G_2$.

The first phase is very important in the generation of the individual, as this is also the one that ensures the correctness of the solution represented by the permutation: as all the values of $V_1$ are assigned from the beginning, and as we assumed $|V_2| > |V_1|$, we conclude that all the nodes of $G_1$ will be matched to any of the nodes of $G_2$ in every solution represented by any permutation.

Therefore, this permutation-based representation is suitable to be used for our problem. The procedure described in this section is shown as pseudocode in Figure 12.2.

## 3.2 Example

To demonstrate the representation of individuals containing permutations and the procedure for translating them to a point in the search space, we consider the example shown in Figure 12.3. In this example we are considering an inexact graph matching problem with a data graph $G_2$ of 10 nodes ($|V_2| = 10$) and a model graph $G_1$ of 6 nodes ($|V_1| = 6$). We also use a similarity measure for the example (the $\varpi(i,j)$ function), the results of which are shown in the same figure. This similarity function does not always have to be symmetrical, and in this example we are using a non-symmetrical one (see Section 3.3 for a discussion on this topic). The translation has to produce individuals of the same size (10 nodes), but each of their values may contain a value between 1 and 6, that is, the number of the node of $V_1$ with which the node of $G_2$ is matched in the solution.

Figure 12.2 shows the procedure for both phases 1 and 2. Following the procedure for phase 1, the first 6 nodes will be matched, and we will obtain the first matches for the three individuals in Figure 12.3.

In the second phase, generation of the solution will be completed by processing one by one all the remaining variables of the individual. For that, we will chose the next variable that is still not treated, the $7^{th}$ in our example. Here, the first individual in the example has the value 7 in its $7^{th}$ position, which means that node 7 of $G_2$ will be worked on next. Similarly, the nodes of $G_2$ to be assigned to the $7^{th}$ position for the other two example individuals are nodes 10 and 4 respectively.

Next, in order to calculate the node of $G_1$ that we have to assign to our node of $G_2$ in the matching, we compare the nodes of $V_2$ that appear before the $7^{th}$ variable in the individual with it. Therefore for the first individual, we compare the similarity between $G_2$ node 7 and each of the $G_2$ nodes 1 to 6. This similarity measure is given by the function $\varpi$ shown in Figure 12.3. If we look at the $7^{th}$ line in this table we see that in columns 1 to 6, the highest value is 0.96, in column 2. Therefore, following the algorithm in phase 2, we assign to node 7 the same matched value as for node 2. As we can see in Figure 12.4, for the first individual, node 2 was assigned the value 2, therefore we will also assign the value 2 to the $7^{th}$ node of $G_2$.

Similarly, for the second individual, the $7^{th}$ variable of the individual is also processed. This has the value 10, so node 10 of $G_2$ is therefore the next to be matched. We will compare this node with the values of the previously matched nodes, i.e. nodes 5, 8, 7, 1, 6 and 9. The highest similarity value for these is $\varpi = 0.97$, in column 9. Therefore the most similar node is node 9, and

Individuals:

| 1 | 2 | 3 | 4 | 5 | 6 ‖ 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|

| 5 | 8 | 7 | 1 | 6 | 9 ‖ 10 | 3 | 4 | 2 |
|---|---|---|---|---|---|---|---|---|

| 10 | 9 | 8 | 7 | 6 | 5 ‖ 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|

Similarity Function:

| $\varpi(i,j)$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1.00 | 0.87 | 0.67 | 0.80 | 0.77 | 0.48 | 0.88 | 0.80 | 0.75 | 0.89 |
| 2 | 0.03 | 1.00 | 0.96 | 0.13 | 0.73 | 0.90 | 0.15 | 0.66 | 0.74 | 0.92 |
| 3 | 0.20 | 0.42 | 1.00 | 0.63 | 0.05 | 0.22 | 0.20 | 0.51 | 0.31 | 0.50 |
| 4 | 0.52 | 0.50 | 0.88 | 1.00 | 0.49 | 0.88 | 0.08 | 0.91 | 0.38 | 0.47 |
| 5 | 0.19 | 0.90 | 0.85 | 0.71 | 1.00 | 0.15 | 0.24 | 0.51 | 0.97 | 0.80 |
| 6 | 0.47 | 0.87 | 0.67 | 0.80 | 0.77 | 1.00 | 0.88 | 0.80 | 0.75 | 0.87 |
| 7 | 0.03 | 0.96 | 0.35 | 0.13 | 0.73 | 0.90 | 1.00 | 0.66 | 0.74 | 0.92 |
| 8 | 0.20 | 0.42 | 0.93 | 0.63 | 0.05 | 0.22 | 0.20 | 1.00 | 0.31 | 0.50 |
| 9 | 0.52 | 0.50 | 0.89 | 0.53 | 0.49 | 0.88 | 0.08 | 0.91 | 1.00 | 0.47 |
| 10 | 0.19 | 0.90 | 0.85 | 0.71 | 0.18 | 0.15 | 0.24 | 0.51 | 0.97 | 1.00 |

*Figure 12.3* Example of three permutation-based individuals and a similarity measure $\varpi(i,j)$ between nodes of the data graph ($\forall i,j \in V_2$) for a data graph of 10 nodes $|V_2| = 10$.

node 10 of $G_2$ will be matched to the same node of $G_1$ as node 9 of $G_2$ was. Looking at Figure 12.4, this is $6^{th}$ node of $G_1$. Following the same process for the third individual, we obtain that node 4 of $G_2$ is matched with node 3 of $G_1$. Figure 12.5 shows the result of this first step of phase 2.

Continuing this procedure of phase 2 until the last variable, we obtain the solutions shown in Figure 12.6.

Note that each of the nodes of $G_2$ is assigned to a variable between 1 and $|V_1| = 6$. Note also that every node of $G_1$ is matched to at least one node of $G_2$, and that a value is given to every node of $G_2$, giving a matching value to each of the segments in the data image (all the segments in the data image are therefore recognised with a structure of the model).

| 1 | 2 | 3 | 4 | 5 | 6 | - | - | - | - |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

| 4 | - | - | - | 1 | 5 | 3 | 2 | 6 | - |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

| - | - | - | - | 6 | 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

*Figure 12.4* Result of the generation of the individual after the completion of phase 1 for the example in Figure 12.3 where six nodes of $G_2$ have been matched ($|V_1| = 6$).

| 1 | 2 | 3 | 4 | 5 | 6 | 2 | - | - | - |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

| 4 | - | - | - | 1 | 5 | 3 | 2 | 6 | 6 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

| - | - | - | 3 | 6 | 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

*Figure 12.5* Generation of the solutions for the example individuals in Figure 12.3 after the first step of phase 2 ($|V_1| = 6$).

An important aspect of this individual representation based on permutations is that the cardinality of the search space is $n!$. This cardinality is higher than that of the traditional individual representation. It is tested for its use with EDAs in graph matching for the first time here. In addition, it is important to note that a permutation-based approach can create redundancies in the solutions, as two different permutations may correspond to the same solution. An example of this is shown in Figure 12.7, where two individuals with different permutations are shown and the solution they represent is exactly the same.

| 1 | 2 | 3 | 4 | 5 | 6 | 2 | 3 | 3 | 3 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

| 4 | 2 | 2 | 2 | 1 | 5 | 3 | 2 | 6 | 6 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

| 1 | 3 | 3 | 3 | 6 | 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

*Figure 12.6* Result of the generation of the solutions after the completion of phase 2.

Individual 1:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|

Individual 2:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 9 | 8 | 10 |
|---|---|---|---|---|---|---|---|---|---|

Solution they represent:

| 1 | 2 | 3 | 4 | 5 | 6 | 2 | 3 | 3 | 3 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

*Figure 12.7* Example of redundancy in the permutation-based approach. The two individuals represent the same solution shown at the bottom of the figure.

## 3.3 Defining the similarity concept

There are three important aspects to consider in order to define the similarity function $\varpi$ for phase 2:

- The first is to decide which nodes have to be compared. In the example we propose comparing nodes from the same graph $G_2$, that is, the model graph $G_1$ has not been taken into account. Other approaches could be considered for instance, taking into account the similarity of both nodes of $G_1$ and nodes of $G_2$ and assigning a weight to both values, or having a

fitness function capable of returning a value for individuals that are not complete.

- Another additional procedure depending on the graph matching problem to be solved is the recalculation of the similarity measure as the individual is being generated: the similarity value could be changed as nodes of the individual are being matched, by following a clustering procedure. This means that in phase 2 an extra clustering procedure would be required in order to update the function $\varpi$.

- And finally, the other aspect to take into account is the definition of the similarity itself. This factor depends on the problem. This definition will determine to an important degree the behavior of the algorithm.

# 4. Obtaining a permutation with discrete EDAs

After describing how permutations can be used in graph matching to obtain correct solutions, the next step is to apply EDAs to this new type of individuals in order to look for the permutation that symbolizes the solution with the optimum fitness value. At the first glance the problem seems a simple application of any EDAs, applying the method described in Section 3.1.

## 4.1 On EDAs applied to graph matching

We will define now more formally the graph matching problem and the way of facing it with an EDA approach, based on the general notation introduced in Chapter 3.

We call $G_1 = (V_1, E_1)$ the model graph and $G_2 = (V_2, E_2)$ the data graph. $V_i$ is the set of nodes and $E_i$ is the set of arcs of graph $G_i$ $(i = 1, 2)$. We still assume that $G_2$ contains more segments than $G_1$. The graph matching task is accomplished by matching nodes from $G_2$ with the nodes of the model graph $G_1$.

We use a permutation as the representation of individuals, which means that the size of these individuals will be of $n = |V_2|$ variables (that is, each individual can be written $\boldsymbol{x} = (x_1, \ldots, x_{|V_2|})$), and each of the $x_i$ can have $|V_2|$ possible values.

## 4.2 Looking for correct individuals

The simulation of Bayesian networks has been used to reason with networks as an alternative to exact propagation methods. In EDAs simulation is used to create the individuals of the following generation based on the structure learned previously.

Among the various methods to perform the simulation process, for this chapter the method of choice is the *Probabilistic Logic Sampling* (PLS) proposed in Henrion (1988).

Nevertheless as explained in Section 2.1, whatever the representation of individuals selected, it is important to check that each individual is correct and satisfies all the restrictions to the problem so that it can be considered as a point in the search space. The interested reader can find a more exhaustive review of this topic in Bengoetxea et al. (2000a), where the authors propose different methods to obtain only correct individuals that satisfy the particular constraints of the problem. In the latter reference two methods to control the simulation step in EDAs are introduced: Last Time Manipulation (LTM) and All Time Manipulation (ATM).

Both methods are based on the modification of the simulation step so that during the simulation of each individual the probabilities learned from the Bayesian network are modified. Each individual is generated variable by variable following the ancestral ordering as in PLS, but the constraints are verified during the instantiation and the probabilities obtained from the learning are modified if necessary to ensure the correctness of the individual.

It is important to note that altering the probabilities at the simulation step, whichever the way, implies that the result of the algorithm is also modified somehow.

For our concrete case of a permutation-based representation, and in order to lead EDAs to the generation of correct permutations only, any of these two methods can be used, and both LTM and ATM will behave exactly in the same way: the only difference between them is that LTM only interacts in the simulation step when the number of values still not appeared equals the number of variables to be simulated in the individual, and that ATM interacts in the probabilities always. As in this case this situation will happen for all the variables of all the individuals, both methods behave in the same way, ensuring in both cases that every possible individual will contain always correct permutations.

## 4.3 Choosing the best discrete EDA algorithm

In order to test EDAs in the inexact graph matching problem defined above, three different EDAs were tested. Typical graph matching problems can have large complexity, and as the difference in behavior between EDAs is to a large extent due to the complexity of the probabilistic structure that they have to build, these three algorithms have been chosen as representatives of the three categories of EDAs introduced in Chapter 3: (1) UMDA (Mühlenbein, 1998) as an example of an EDA that considers no interdependencies between the variables; (2) MIMIC (De Bonet et al., 1997) is an example that belongs to

the category of pairwise dependencies; (3) EBNA (Etxeberria and Larrañaga, 1999) multiple interdependencies are allowed.

## 5. Obtaining a permutation with continuous EDAs

Continuous EDAs provide the search with other types of EDAs that can be more suitable for some problems. But again, the main goal is to find a representation of individuals and a procedure to obtain an univocal solution to the matching from each of the possible permutations.

In this case we propose a strategy based on the previous section, trying to translate the individual in the continuous domain to a correct permutation in the discrete domain, proceeding next as explained in Section 3.1.

This procedure of translating from the continuous world to the discrete world has to be performed for each individual in order to be evaluated. Again, this process has to be fast enough in order to reduce computation time.

With all these aspects in mind, individuals of size $n = |V_2|$ will be defined. Each individual is obtained sampling from a n-dimensional Gaussian distribution, and therefore can take any value in $\mathbb{R}^n$. With this new representation the individuals do not have a direct meaning of the solution it represents: the values for each of the variables do only show the way to translate from the continuous world to a permutation as with the discrete representation shown in Section 2.1, and it does not contain similarity values between nodes of any graph. This new type of representation can also be regarded as a way of change the search from the discrete to the continuous world, where the techniques that can be applied to the estimation of densities are completely different.

To obtain a translation to a discrete permutation, we order the continuous values of the individual, and set its corresponding discrete values by assigning to each $x_i \in \{1, \ldots, |V_2|\}$ the respective order. The procedure described in this section is shown as pseudocode in Figure 12.8.

For the simulation of an univariate normal distribution, a simple method based on the sum of 12 uniform variables (Box and Muller, 1958) is chosen. On the other hand, the sampling of multivariate normal distributions has been done by means of an adaptation of the conditioning method (Ripley, 1987) on the basis of the PLS algorithm. Note that in this continuous case it is not required to check whether all the values are different or not.

Again, for the continuous domain different EDAs are proposed and these are to be tested in this chapter for their performance in a concrete inexact graph matching problem. Three different algorithms are chosen again, as representatives of their complexity category. These are the $\text{UMDA}_c$, $\text{MIMIC}_c$, and EGNA (Larrañaga et al., 2000).

**From a continuous value in $\mathbb{R}^n$ to a discrete permutation**

**Definitions**

$n = |V_2|$: size of the individual, which is the number of nodes in data graph $G_2$ (the permutation)

$\boldsymbol{x^C} = (x_1^C, \ldots, x_{|V_2|}^C)$: individual containing continuous values (the input)

$\boldsymbol{x^D} = (x_1^D, \ldots, x_{|V_2|}^D)$: individual containing a permutation of discrete values (the output)

$x_i^D \in \{1, \ldots, n\}$: value of the $i^{th}$ variable in the individual

**Procedure**

Order the values $x_1^C, \ldots, x_{|V_2|}^C$ of individual $\boldsymbol{x^C}$ using any fast sorting algorithm such as Quicksort

Let $K_i$ be position in which each value $x_i^C$, $1 \leq i \leq |V_2|$, occupies after ordering all the values

The values of the individual $\boldsymbol{x^D}$ will be set in the following way: $\forall\, i = 1, \ldots, |V_2|$, $x_i^D = K_i$

*Figure 12.8* Pseudocode to translate from a continuous value in $\mathbb{R}^n$ to a discrete permutation composed of discrete values.

# 6. Experimental results. The human brain example

## 6.1 Overview of the human brain example

The example chosen to test the new permutation-based representation is an inexact graph matching one used for recognition of structures in Magnetic Resonance Images (MRI). The data graph $G_2$ is generated from this image and contains a node for each region (subset of a brain structure). The model graph $G_1$ is built from an anatomical atlas and each node corresponds exactly to one brain structure. The experiments carried out in this chapter are focused on this type of graphs, but could similarly be adapted to any other inexact graph matching problem.

More specifically, the model graph has been obtained from the main structures of the brainstem, the inner part of the brain, and it does not take into account the cerebral hemispheres. This reduced example is a shorter version of the brain images recognition problem in Perchant and Bloch (1999). The

fact that less structures have to be recognized (from 43 to 12) reduces the complexity of the problem. In the same way, the human brain images have also been reduced, and the number of structures of the data image to be matched (number of nodes of $G_2$) is also reduced from 245 to 94. The number of arcs is also different in this problem: while in Perchant and Bloch (1999) $G_1$ and $G_2$ contained 417 and 1451 arcs, in these examples the number of arcs is of 84 and 2868 respectively.

Speaking about the similarity concept, for our experiments we have used only a similarity measure based on the grey level distribution, so that when the function $\varpi$ returns a higher value for two nodes it shows a more similar grey level distribution over the two segments of the data image. Another possible property could have been the distance between the segments in the data image for instance. In addition, no extra computation is performed during the generation of the individual (not clustering process is performed), and therefore the similarity measure $\varpi$ is kept as a constant during the generation of individuals. These decisions have been taken knowing the nature and properties of the data graph, which is a human brain NMR image in black and white. These decisions were also considered as a way to simplify the complexity of the problem.

## 6.2 Description of the experiment

The aim of these experiments is to test the performance of some discrete and continuous EDAs introduced in Chapter 3 in this volume for the same example. As the main difference between them is the number of dependencies between variables that they take into account, the more complex algorithms are expected to require more CPU time but also to reach a fitter final solution. This section describes the experiments and the results obtained. EDAs are also compared to a broadly known GA, the GENITOR (Whitley and Kauth, 1988), which is a steady state type algorithm (ssGA) (Michalewicz, 1992).

Both EDAs and GENITOR were implemented in ANSI C++ language, and the experiment was executed in a two processor Ultra 80 Sun computer under Solaris version 7 with 1 Gb of RAM.

In the discrete case, all the algorithms were designed to end the search when a maximum of 100 generations or when uniformity in the population was reached. GENITOR is a special case, as it is a ssGA and therefore generates only one individual at each iteration, but it was also programmed in order to generate the same number of individuals as in discrete EDAs by allowing more iterations (201900 individuals). In the continuous case, the ending criterion was to reach 301850 evaluations (i.e. number of individuals generated).

The initial population for all the algorithms was generated using the same random generation procedure based on a uniform distribution. The fitness function used is described later in Section 6.3.

In EDAs, the following parameters were used: a population of 2000 individuals ($M = 2000$), from which a subset of the best 1000 are selected ($N = 1000$) to estimate the probability, and the elitist approach was chosen (that is, always the best individual is included for the next population and 1999 individuals are simulated). In GENITOR a population of 2000 individuals was also set, with a mutation rate of $p_m = \frac{1}{|V_2|}$ and a crossover probability of $p_c = 1$. The operators used in GENITOR where CX (Oliver et al., 1987) and EM (Banzhaf, 1990).

## 6.3 Definition of the fitness function

The definition of the fitness function for the graph matching problem will be a very important factor in the resolution of the problem as well, as its behavior will also determine how the optimization algorithm approaches the best solution. It is important to define appropriately the function that will be used in order to compare individuals and obtain the best solution. The aim of this chapter is not to do a review of the different fitness functions for graph matching. This is the reason why the function proposed in Perchant and Bloch (1999) will be used just as an example of a fitness function in inexact graph matching. This function has been used to solve the problem applied to human brain images with GAs in Perchant et al. (1999) and Boeres et al. (1999) and with EDAs in Bengoetxea et al. (2000a) and Bengoetxea et al. (2000b). Following this function, an individual $\boldsymbol{x} = (x_1, \ldots, x_{|V_2|})$ will be evaluated as follows:

$$f(\boldsymbol{x}; \rho_\sigma, \rho_\mu, \alpha) = \alpha \left[ \frac{1}{|V_2||V_1|} \sum_{i=1}^{|V_2|} \sum_{j=1}^{|V_1|} \left(1 - |c_{ij} - \rho_\sigma^{u_1^i}(u_2^j)|\right) \right] +$$

$$(1-\alpha) \left[ \frac{1}{|E_2||E_1|} \sum_{e_1^l = (u_1^i, v_1^{i'}) \in E_1} \sum_{e_2^k = (u_2^j, v_2^{j'}) \in E_2} \left(1 - |c_{ij} c_{i'j'} - \rho_\mu^{e_1^l}(e_2^k)|\right) \right] \quad (12.3)$$

where

$$c_{ij} = \begin{cases} 1 & \text{if } x_i = j \\ 0 & \text{otherwise} \end{cases}$$

and $\alpha$ is a parameter used to adapt the weight of node and arc correspondences in $f$, and $\rho_\sigma = \{\rho_\sigma^{u_1} : V_2 \to [0,1], u_1 \in V_1\}$ is the set of functions that measure the correspondence between the nodes of both graphs $G_1$ and $G_2$. Similarly, $\rho_\mu = \{\rho_\mu^{(u_1,v_1)} : E_2 \to [0,1], (u_1, v_1) \in E_1\}$ is the set of functions that measure the correspondence between the arcs of both graphs $G_1$ and $G_2$. The value of $f$ associated for each variable returns the goodness of the matching. Typically $\rho_\sigma$ and $\rho_\mu$ are related to the similarities between node properties and arc properties respectively.

Function $f(\boldsymbol{x}; \rho_\sigma, \rho_\mu, \alpha)$ has to be maximized.

*Table 12.1* Mean values of experimental results after 10 executions for each algorithm of the inexact graph matching problem of the Human Brain example.

| | *Best fitness value* | *Execution time* | *Number of evaluations* |
|---|---|---|---|
| UMDA | 0.718623 | 00:53:29 | 85958 |
| $UMDA_c$ | 0.745036 | 03:01:05 | 301850 |
| MIMIC | 0.702707 | 00:57:30 | 83179 |
| $MIMIC_c$ | 0.747970 | 03:01:07 | 301850 |
| EBNA | 0.716723 | 01:50:39 | 85958 |
| EGNA | 0.746893 | 04:13:39 | 301850 |
| ssGA | 0.693575 | 07:31:26 | 201900 |
| | $p < 0.001$ | $p < 0.001$ | $p < 0.001$ |

## 6.4 Experimental results

Results such as the best individual obtained, the computation time, and the number of evaluations to reach the final solution were recorded for each of the experiments.

The computation time is the CPU time of the process for each execution, and therefore it is not dependent on the multiprogramming level at the moment of the execution. This computation time is presented as a measure to illustrate the different computation complexity of all the algorithms. It is important also to note that all the operations for the estimation of the distribution, the simulation, and the evaluation of the new individuals are carried out through memory operations.

Each algorithm was executed 10 times, and the null hypothesis of the same distribution densities was tested for each of them. The non-parametric tests of Kruskal-Wallis and Mann-Whitney were used. This task was carried out with the statistical package S.P.S.S. release 9.00 and the results are shown in Table 12.1.

This table shows the mean results for each of the experiments, showing the different parameters (best fitness value obtained, execution time and number of generations required respectively). Additionally, the same Kruskal-Wallis and Mann-Whitney tests were also applied to test the differences between particular algorithms. The results were as follows:

- Between algorithms of similar complexity only:
  - UMDA vs. $UMDA_c$. Fitness value: $p < 0.001$; CPU time: $p < 0.001$; Evaluations: $p < 0.001$.

- MIMIC vs. MIMIC$_c$. Fitness value: $p < 0.001$; CPU time: $p < 0.001$; Evaluations: $p < 0.001$.
- EBNA vs. EGNA. Fitness value: $p < 0.001$; CPU time: $p < 0.001$; Evaluations: $p < 0.001$.

From the results we can conclude that the differences between the algorithms in the discrete and continuous domains are significant for all the algorithms analyzed. This means that the behaviour of selecting a discrete learning algorithm or its equivalent in the continuous domain is very different regarding all the parameters analyzed. It is important to note that the number of evaluations was expected to be different, as the ending criteria for the discrete and continuous domains have been set to be different. In all the cases, the continuous algorithms obtained a fitter individual, but the CPU time and number of individuals created was also bigger.

- Between discrete EDAs only:
  - Fitness value: $p < 0.001$.
  - CPU time: $p < 0.001$.
  - Evaluations: $p < 0.001$.

  In this case significant results are also obtained in fitness value and CPU times, as well as in the number of evaluations. The discrete algorithm that obtained the best result was UMDA, closely followed by EBNA. The differences in the CPU time are also according to the complexity of the learning algorithm we used. Finally, the different number of evaluations means that MIMIC required significantly less individuals to converge (to reach the uniformity in the population), whereas the other two EDAs require quite the same number of evaluations to converge.

  The genetic algorithm GENITOR is far behind the performance of EDAs. The computation time is also a factor to be taken into account: the fact that GENITOR requires about 7 hours for each execution can give an idea of the complexity of the problem that these algorithms are dealing with.

- Between continuous EDAs only:
  - Fitness value: $p = 0.342$.
  - CPU time: $p < 0.001$.
  - Evaluations: $p = 1.000$.

  In the case of the continuous algorithms, the differences in fitness value between the different learning methods are not significant in the light of

the results. Nevertheless, the CPU time required for each of them is also according to the complexity of the learning algorithm. On the other hand, as the ending criterion for all the continuous algorithms was to reach the same number of evaluations, it was obvious that there were not differences between them in the number of evaluations. Speaking about the differences in computation time between discrete and continuous EDAs, it is important to note that the latter ones require all the 301850 individuals to be generated before they finish the search. Furthermore, the computation time for the continuous algorithms is also longer than the discrete equivalents as a result of several factors: firstly, due to the higher number of evaluations they perform each execution, secondly because of the longer individual-to-solution translation procedure that has to be done for each of the individuals generated, and lastly, as a result of the longer time required to learn the model in continuous spaces.

In the light of the results obtained in the fitness values, we can conclude the following: generally speaking, continuous algorithms perform better than discrete ones, either when comparing all of them in general or when only with algorithms of equivalent complexity.

## 7. Conclusions and further work

This chapter introduces a new individual representation approach for EDAs applied to the inexact graph matching problem. This new individual representation can be applied in both discrete and continuous domains.

In experiments carried out with a real example, a comparison of the performance of this new approach between the discrete and continuous domains has been done, and continuous EDAs have shown a better performance looking at the fittest individual obtained, however a longer execution time and more evaluations were required. Additionally, other fitness functions should be tested with this new approach. Techniques such as Bloch (1999a) and Bloch (1999b) could also help to introduce better similarity measures and therefore improve the results obtained considerably.

## Acknowledgments

This chapter has been partially supported by the Spanish Ministry for Science and Education, and the French Ministry for Education, Research and Technology with the projects HF1999-0107, and Picasso-00773TE respectively. The authors would also like to thank R. Etxeberria, I. Inza and J.A. Lozano for their useful advice and contributions to this work.

## References

Banzhaf, W. (1990). The molecular traveling salesman. *Biological Cybernetics*, 64:7–14.

Bengoetxea, E., Larrañaga, P., Bloch, I., Perchant, A., and Boeres, C. (2000a). Inexact graph matching using learning and simulation of Bayesian networks. An empirical comparison between different approaches with synthetic data. In *Proceedings of CaNew workshop, ECAI 2000 Conference, ECCAI*, Berlin.

Bengoetxea, E., Larrañaga, P., Bloch, I., Perchant, A., and Boeres, C. (2000b). Learning and simulation of Bayesian networks applied to inexact graph matching. *International Journal of Approximate Reasoning.* (submitted).

Bloch, I. (1999a). Fuzzy relative position between objects in image processing: a morphological approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(7):657–664.

Bloch, I. (1999b). On fuzzy distances and their use in image processing under imprecision. *Pattern Recognition*, 32:1873–1895.

Boeres, C., Perchant, A., Bloch, I., and Roux, M. (1999). A genetic algorithm for brain image recognition using graph non-bijective correspondence. Unpublished manuscript.

Box, G.E.P. and Muller, M.E. (1958). A note on the generation of random normal deviates. *Ann. Math. Statist.*, 29:610–611.

Cross, A.D.J. and Hancock, E.R. (1998). Graph matching with a dual-step EM algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(11):1236–53.

Cross, A.D.J. and Hancock, E.R. (1999). Convergence of a hill climbing genetic algorithm for graph matching. In Hancock, E.R. and Pelillo, M., editors, *Lectures Notes in Computer Science 1654*, pages 220–236, York, UK.

Cross, A.D.J., Wilson, R.C., and Hancock, E.R. (1997). Inexact graph matching using genetic search. *Pattern Recognition*, 30(6):953–70.

De Bonet, J.S., Isbell, C.L., and Viola, P. (1997). MIMIC: Finding optima by estimating probability densities. *Advances in Neural Information Processing Systems*, Vol. 9.

Etxeberria, R. and Larrañaga, P. (1999). Global optimization with Bayesian networks. In *II Symposium on Artificial Intelligence. CIMAF99. Special Session on Distributions and Evolutionary Optimization*, pages 332–339.

Finch, A.W., Wilson, R.C., and Hancock, E.R. (1997). Matching Delaunay graphs. *Pattern Recognition*, 30(1):123–40.

Finch, A.W., Wilson, R.C., and Hancock, E.R. (1998). Symbolic graph matching with the EM algorithm. *Pattern Recognition*, 31(11):1777–90.

Gold, S. and Rangarajan, A. (1996). A graduated assignment algorithm for graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(4):377–88.

Hancock, E.R. and Kittler, J. (1990). Edge-labeling using dictionary-based relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(2):165–181.

Henrion, M. (1988). Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In Lemmer, J.F. and Kanal, L.N., editors, *Uncertainty in Artificial Intelligence*, volume 2, pages 149–163. North-Holland, Amsterdam.

Larrañaga, P., Etxeberria, R., Lozano, J.A., and Peña, J.M. (2000). Optimization in continuous domains by learning and simulation of Gaussian networks. In *Proceedings of the Workshop in Optimization by Building and using Probabilistic Models. A Workshop within the 2000 Genetic and Evolutionary Computation Conference, GECCO 2000*, pages 201–204, Las Vegas, Nevada, USA.

Lovàsz, L. and Plummer, M.D. (1986). *Matching Theory.* Mathematics Studies. Elsevier Science, North-Holland.

Michalewicz, Z. (1992). *Genetic algorithms + data structures = Evolution Programs.* Springer Verlag, Berlin Heidelberg.

Mühlenbein, H. (1998). The equation for response to selection and its use for prediction. *Evolutionary Computation*, 5:303–346.

Myers, R. and Hancock, E.R. (2001). Least committment graph matching with genetic algorithms. *Pattern Recognition*, 34:375–394.

Oliver, J., Smith, D., and Holland, J. (1987). A study of permutation crossover operators on the TSP. In Grefenstette, J.J., editor, *Proceedings of the Second International Conference on Genetic Algorithms and Their Applications*, pages 224-230. Lawrence Erlbaum Associates.

Perchant, A. (2000). *Morphism of graphs with fuzzy attributes for the recognition of structural scenes.* PhD Thesis, Ecole Nationale Supérieure des Télécommunications, Paris, France (In french).

Perchant, A. and Bloch, I. (1999). A New Definition for Fuzzy Attributed Graph Homomorphism with Application to Structural Shape Recognition in Brain Imaging. In *IMTC'99, 16th IEEE Instrumentation and Measurement Technology Conference*, pages 1801–1806, Venice, Italy.

Perchant, A., Boeres, C., Bloch, I., Roux, M., and Ribeiro, C. (1999). Model-based Scene Recognition Using Graph Fuzzy Homomorphism Solved by Genetic Algorithms. In *GbR'99 2nd International Workshop on Graph-Based Representations in Pattern Recognition*, pages 61–70, Castle of Haindorf, Austria.

Ripley, B.D. (1987). *Stochastic Simulation.* John Wiley and Sons.

Singh, M. and Chaudhury, A.C.S. (1997). Matching structural shape descriptions using genetic algorithms. *Pattern Recognition*, 30(9):1451–62.

Whitley, D. and Kauth, J. (1988). GENITOR: A different genetic algorithm. In *Proceedings of the Rocky Mountain Conference on Artificial Intelligence*, volume 2, pages 118–130.

Wilson, R.C. and Hancock, E.R. (1996). Bayesian compatibility model for graph matching. *Pattern Recognition Letters*, 17:263–276.

Wilson, R.C. and Hancock, E.R. (1997). Structural matching by discrete relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(6):634–648.