

CONTEXT-SENSITIVE MATCHING OF TWO SHAPES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

EMRE BAŞESKİ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR

THE DEGREE OF MASTER OF SCIENCE

IN

COMPUTER ENGINEERING

JULY 2006

Approval of the Graduate School of Natural and Applied Sciences.

Prof. Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Ayşe Kiper
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Sibel Tari
Supervisor

Examining Committee Members

Prof. Dr. Neşe Yalabık (METU,CENG)_____

Assoc. Prof. Dr. Sibel Tari (METU,CENG)_____

Prof. Dr. Volkan Atalay (METU,CENG)_____

Assoc. Prof. Dr. İsmail Hakkı Toroslu (METU,CENG)_____

Assist. Prof. Dr. Mine Özkar (METU,ARCH)_____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name:

Signature:

ABSTRACT

CONTEXT-SENSITIVE MATCHING OF TWO SHAPES

Bağeski, Emre

M.Sc., Department of Computer Engineering

Supervisor: Assoc. Prof. Dr. Sibel Tari

July 2006, 70 pages

The similarity between two shapes is typically calculated by measuring how well the properties and the spatial organization of the primitives forming the shapes agree. But, when this calculations are done independent from the context, i.e. the whole set of shapes in the experiments, a priori significance to the primitives is assigned, which may cause problematic similarity measures. A possible way of using context information in similarity measure between shape A and shape B is using the category information of shape B in calculations. In this study, shapes are represented as depth-1 shape trees and the dissimilarity between two shapes is computed by using an approximate tree matching algorithm. The category information is created as the union of shape trees that are in the same category and this information guides the matching process between a query shape and a shape whose category is known.

Keywords: tree editing distance, shape matching.

ÖZ

İKİ ŞEKLİN BAĞLAMA DUYARLI KARŞILAŞTIRILMASI

Başeski, Emre

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Assoc. Prof. Dr. Sibel Tari

Temmuz 2006, 70 sayfa

İki şekil arasındaki benzerlik, tipik olarak şekilleri oluşturan yapıtaşlarının özellikleri ve uzaysal organizasyonlarının ne kadar uyumlu olduğunu ölçerek hesaplanır. Fakat, bu hesaplamalar bağlamdan, örneğin deneylerin yapıldığı tüm şekillerin kümesinden, bağımsız yapılırsa yapıtaşlarına fazla önem atanır ki bu da problematik benzerlik ölçütlerine sebebiyet verebilir. A ve B şekilleri arasındaki benzerlik ölçütünde bağlam bilgisini kullanmanın muhtemel bir yolu, B şeklinin kategori bilgisini hesaplarda kullanmaktır. Bu çalışmada, şekiller 1 derinlikli şekil ağaçları olarak betimlenmektedir ve iki şekil arasındaki benzerlik, yaklaşımsal bir ağaç karşılaştırma algoritması ile hesaplanmaktadır. Kategori bilgisi, aynı kategorideki şekil ağaçlarının birleştirilmesi ile yaratılmaktadır ve bu bilgi, sorgulanan bir şekil ve kategorisi bilinen bir şekil arasındaki karşılaştırma işlemine yol göstermektedir.

Anahtar Kelimeler: ağaç değiştirme mesafesi, şekil karşılaştırma.

To my parents

ACKNOWLEDGMENTS

I would like to thank to Sibel Tari for her supervision, guidance and invaluable suggestions throughout the development of this thesis. I would also like to thank Aykut Erdem for his great friendship and sharing hard times of my work.

TABLE OF CONTENTS

PLAGIARISM	iii
ABSTRACT	iv
ÖZ	v
DEDICATON	vi
ACKNOWLEDGMENTS	vii
TABLE OF CONTENTS	viii
LIST OF TABLES	x
LIST OF FIGURES	xi
CHAPTER	
1 INTRODUCTION	1
2 REVIEW OF DISCONNECTED SKELETON	6
2.1 Disconnected Skeleton	6
2.2 Canonical Coordinate Frame	8
3 FROM DISCONNECTED SKELETON TO SHAPE TREE	10
3.1 Construction of Ordered Attributed Depth-1 Shape Tree from Symmetry Branches	10
3.1.1 Attributes of a Node in the Shape Tree	11
3.2 Sample Shape Trees	13
4 APPROXIMATE TREE MATCHING	17
4.1 Tree Editing Distance Algorithm of Shasha and Zhang	18
4.2 Mapping Algorithm	22

5	SHAPE MATCHING USING TREE EDITING DISTANCE . . .	25
5.1	Calculation of Dissimilarity Measure	26
5.1.1	Insertion and Deletion Costs	26
5.1.2	Attribute Change Cost	27
5.2	Mapping	28
5.3	Experiments	29
5.3.1	Clustering Results	30
5.4	Discussions	31
5.4.1	Structurally Different Shapes with Similar Shape Trees	32
5.4.2	Structurally Similar Shapes	33
5.4.3	Shapes with Extra Branches	33
5.4.4	Outliers in Distribution of Lengths of Branches	34
6	CONTEXT SENSITIVE MATCHING	46
6.1	Construction of Shape-Category Tree	47
6.2	Using Shape-Category Trees In Matching	51
6.2.1	Context Guided Editing Operation Costs . . .	52
6.3	Experiments	53
7	CONCLUSION	67
	REFERENCES	68

LIST OF TABLES

3.1	Sample Skeleton to Tree Conversions	12
3.2	Sample skeletons from each class in shape database	14
5.1	Retrieval results	35
6.1	Normalized lengths of tails of crocodiles	54
6.2	Retrieval results	56

LIST OF FIGURES

1.1	An experiment from [1]	3
1.2	The shape database used in the experiments. There are 180 shapes from 30 categories.	4
2.1	Disconnected skeletons from [2, 3]	7
2.2	Triple junction example (a) Shape with mirror symmetry (b)Part of the shape that causes triple junction (c)Part of the symmetry axis: Intersecting branches at junction point (d)-(e) Skeletal configurations which arise as a result of slight deviation from mirror symmetry	8
2.3	(a) Triple junction in the case of perfect mirror symmetry. (b)Triple junction disappear when corners are smoothed.	8
2.4	Four possible reference axes of the hand shape from [2]	9
2.5	A sample coordinate frame from [2]	9
3.1	(a) Disconnected skeleton of an elephant shape. (b)-(c) Two alternative parses.	11
4.1	Edit operations (a)label change (b)delete (c)insert	19
4.2	A sample mapping	19
4.3	$l(5) = 1$ and $l(2) = 2$	20
4.4	$T[2..4]$ of the tree in Figure 4.3	20
5.1	Deleting a long branch may convert the shape to a totally different shape. (a)Original shape (b)Shape tree of a (c) The longest branch in b is deleted	26
5.2	Number of nodes in shape trees of different shapes may be equal	28
5.3	Sample mappings	29
5.4	Mapping of two hand shapes one with a missing finger	29
5.5	Sample shapes to be clustered with NCuts clustering algorithm	30
5.6	Shapes in 5.5 clustered into 3	31
5.7	Shapes in 5.5 clustered into 5	31
5.8	Shapes in 5.5 clustered into 10	32
5.9	Similar skeletons for an elephant shape and a squirrel shape	32
5.10	Similar shapes. A cat shape and a horse shape	33
5.11	Problematic embryo shapes one with more branches	33

5.12	A mismatch because of an outlier. (a)key shape (b)misc5 shape	34
6.1	Shapes in the same category may contain extra branches	47
6.2	A sample shape category	48
6.3	Initial shape-category tree of the category in Figure 6.2	49
6.4	Shape-category tree of the category in Figure 6.2 after adding T_2	49
6.5	Shape-category tree of the category in Figure 6.2 after adding T_3	49
6.6	Category tree for the hand	51
6.7	Using shape-category tree in matching	51
6.8	The cost function $f(x, y, [min, max])$ used in prior-guided matching.	52
6.9	An outlier shape in the category	54
6.10	A shape accepted by another category	54
6.11	Structurally similar shapes from different categories	54

CHAPTER 1

INTRODUCTION

Matching two shapes and measuring their similarity is an important task for vision. Typically, shape similarity is a measure of how well the primitives forming two shapes and their spatial organization agree. Hence the tree data structure provides a very natural way of storing the shape primitives according to their spatial organization (inclusion relation), it is widely used to represent a shape. When a shape (primitives + inclusion relations) is expressed by a tree, the shape dissimilarity can be computed as the tree editing distance. To find the similarity among two shapes using skeletal trees, the basic steps are: *a)* computation of shape primitives, *b)* formation of shape trees, *c)* comparison of shape trees.

Quite a significant majority of tree or graph based shape matching schemes make use of some axial a.k.a skeletal representation to determine the primitives and the inclusion relations [4, 5, 6, 7]. Skeleton extraction is quite a sensitive procedure and one of the most promising developments is the concept of shape scale space by Kimia, Tanenbaum and Zucker. [8]. Alternative formulations are in Tari, Shah and Pien. [9], Latecki and Lakamper [10] and Imiya and Eckhardt [11].

The obtained shape primitives and their spatial organization are used in shape tree formation. In the shock tree of Siddiqi et. al. [4], the center of the largest circular piece is designated as the root and the primitives (pieces

of skeletons) are represented by vertices. In the shape axis tree of Liu and Geiger [5], on the other hand, primitives are represented by the tree edges. A major source of difficulty in tree based approaches is that small changes in a shape can lead to a major reorganization of the tree making the representation, hence, the matching schemes built on this representation very unstable. To make skeletal tree based methods work in practice, various ideas have been developed. An interesting idea is the incorporation of top-down verification process as in [6] to adjust bottom-up skeleton extraction. To compensate for the errors in the skeleton computation, Liu and Geiger [5] consider not only node-to-node correspondences but also node-to-path correspondences. The motivation behind both approaches is to allow multiple parses of the same shape to deal with instabilities of the skeleton extraction.

After shape tree formation, the similarity between two shapes can be found by comparing their shape trees. One alternative is to formulate the partial match computation as a sub-graph isomorphism problem. Another alternative is to formulate it via tree edit operations which transforms one tree to another as in Shasha and Zhang [12]. In such case, the distance between two shapes is measured by the minimum cost of editing which is referred as the tree edit distance. Typically, this cost is symmetric.

In the shape matching literature, similarity is traditionally defined via metrics as such it is symmetric and satisfies triangle inequality, with the exception of most recent works in computer vision, including [1, 13]. Jacobs, Weinshall and Gdalyahu [13] discuss the motivations for using non-metric measures of similarity and gives a good account of examples from the recent literature, focusing on the departure violation of triangle inequality. In Figure 1.1, an experiment from [1] is presented. Hence the observers states that the dissimilarity between person and horse is more than twice as great as the dissimilarity between either shape and the centaur, there is a violation of triangle inequality.

Experimental studies also demonstrates that human similarity judgements are asymmetric [14, 15]. Both Mumford [16] and Edelman et.al. [17] interprets this asymmetry as context dependence. The experiment conducted by Tversky

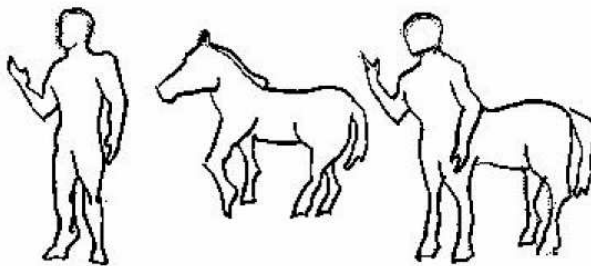


Figure 1.1: An experiment from [1]

[14] shows that, for the discrimination of A,B,C, and D, if C and D are very different from A and B, A and B are often confused.

In this study, the effect of context to the calculation of similarity between two shapes is discussed. The main idea is using the guidance of context to obtain a bias which is used to find a similarity measure. To test the approach, image retrieval problem is studied, where the aim is finding the most similar shapes in an image database for a query shape.

Context sensitive matching is investigated by letting the category descriptions to represent the context information and influence the process of editing one skeletal tree into another. Even though the underlying similarity measure is symmetric, the bias due to assumed category distorts the symmetry. Departure from a symmetry may be very useful in a shape retrieval application, where a query (stimulus input) is compared against a prototype (memory benchmark).

For the experiments, a shape database composed of 180 shapes with 30 categories, collected from various sources [7, 2] is used (see Figure 1.2).

In this thesis, attributed, ordered, depth-1 tree is the main data structure for both category and shape representation. A category tree is sort of a union tree which captures the within category variability of the primitives and the inclusion relations via vector valued attributes attached to nodes and vertices. Forming union of tree representations has been addressed by Torsello and Hancock [18]. Unlike Torsello and Hancock's construction in which union of trees may not result in a tree structure, our category construction always produces a depth-1 tree.

As demonstrated in this work, using depth-1 trees offer both computational

human	star
elephant	tortoise
misc1	dumbbell
misc2	misc3
cat	palm
hand	ship
embryo	turtle
club	flower
misc5	crocodile
squirrel	frog
rabbit	misc6
misc7	horse
umbrella	crown
dino	fish
key	dog

Figure 1.2: The shape database used in the experiments. There are 180 shapes from 30 categories.

and conceptual advantages as:

- tree editing distance computation becomes equivalent to string edit distance computation
- category prototype construction process becomes trivial
- representing both individual shapes and categories with the same data

structure leads to a quite trivial mechanism for introducing bias for shape primitives

- allows individual shape and category comparison via a tree editing distance algorithm

What makes depth-1 trees partially possible is a new axis extraction method of Aslan and Tari [2] which extracts the axes at a very coarse scale at which the shape is perceived as a single blob. He refers to this representation as the *disconnected skeleton*.

Thesis Organization

In Chapter 2, the disconnected skeleton representation of Aslan and Tari [2] is reviewed. In Chapter 3, the methodology that is used to parse a disconnected skeleton into a pair (or more) of ordered depth-1 trees is discussed. An approximate tree matching algorithm is given in Chapter 4. In Chapter 5, the branch and bound matching scheme in [2] is studied from a tree matching perspective and its limitations are explored. Then, by letting the category of second shape to influence the tree-to-tree matching in Chapter 6, the matching problem is examined by using a context, which is referred as *context-sensitive matching*. Finally, in Chapter 7, the presented work is concluded.

Contributions

The contributions of this thesis are: *a)* reducing the algorithmic complexity of the matching scheme presented in [2] by using an approximate tree matching algorithm *b)* calculating dissimilarity between two shapes by paying attention to the context, in which the calculation is performed *c)* allowing shape-category and category-category comparisons by using the shape-category tree.

CHAPTER 2

REVIEW OF DISCONNECTED SKELETON

In this chapter, the disconnected skeleton representation of Aslan and Tari [2] is reviewed. After presenting how disconnected skeletons are computed, the canonical coordinate frame that is used to determine the spatial organization of primitives is discussed.

2.1 Disconnected Skeleton

Consider a shape whose boundary is Γ . Let v be the solution of the following differential equation:

$$\nabla^2 v = \frac{v}{\rho^2}, \quad v|_{\Gamma} = 1 \quad (2.1)$$

As shown in [9], successive level curves of v are gradually smooth versions of Γ and the amount of smoothing increases with increasing ρ . v function is closely related to the Ambrosio-Tortorelli approximation [19] of Mumford-Shah segmentation functional [20]. In [9], this connection is explored and v function is proposed as a multiscale alternative to distance transform which can be directly computed from unsegmented real images. Recently, Aslan et. al. [2] showed that, by selecting a small ρ in Equation (2.1) and increasing it until a function with a single extremum point is obtained, all shapes shrink into a single point. This numerical trick leads to unique shape center for any shape.

Resulting surface v permits to a very stable shape analysis by capturing the ribbon-like sections in the form of a *disconnected skeleton* – a set of disconnected protrusion and indentation branches. Each protrusion branch emanates from a boundary protrusion and ends at an interior point or at the shape center. The interior points are very stable to articulations and boundary perturbations [3] (see Figure 2.1). There are at least two protrusion and two indentation branches which reach to shape center. These branches are called major branches. If the symmetry at the center is more than 2-fold then the number of major branches increase to $n > 2$ to reflect the n -fold symmetry [3].

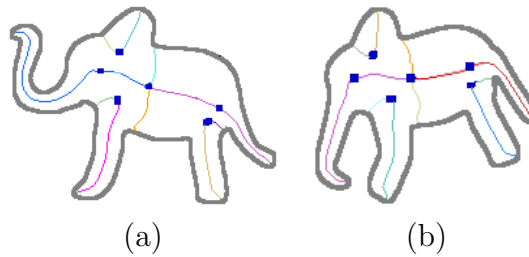


Figure 2.1: Disconnected skeletons from [2, 3]

The evolution of level curves determines the sign of symmetry branches. Symmetry points that track the evolution of the protrusions of a shape form positive symmetry branches and symmetry points that track the evolution of the indentations of a shape form negative symmetry branches.

In case of perfect mirror symmetry about a skeletal branch, three branches (two positive and one negative) may join, forming a *triple junction*. A sample shape with a triple junction is shown in Figure 2.2 (a). The object part that is shown in Figure 2.2 (b) is mirror symmetric and the two positive branches emanating from the two curvature maxima (corners) and the negative branch emanating from the curvature minima intersect and continue as a single positive branch (Figure 2.2 (c)).

Triple junctions are instable configurations. Slight deviation from perfect mirror symmetry will change the axis section shown in Figure 2.2(c) to the either one of the axis sections shown in Figure 2.2(d) and 2.2(e). Triple junctions also disappear when sharp corners are replaced by smooth corners. Consider the sample in Figure 2.3(a). When corners are smoothed (Figure 2.3(b)), the triple

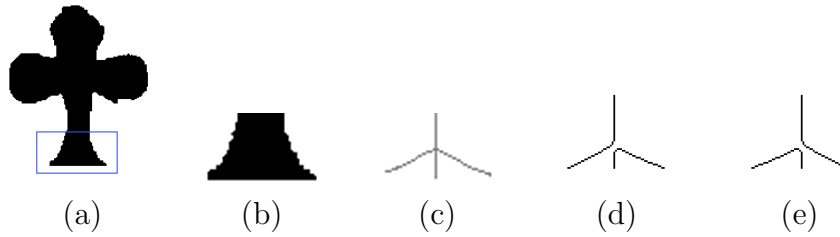


Figure 2.2: Triple junction example (a) Shape with mirror symmetry (b)Part of the shape that causes triple junction (c)Part of the symmetry axis: Intersecting branches at junction point (d)-(e) Skeletal configurations which arise as a result of slight deviation from mirror symmetry

junction disappear. As such, the descriptions shown in Figure 2.2(d)-(e) are more stable.

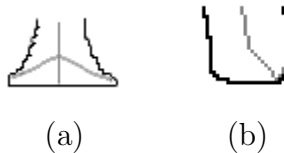


Figure 2.3: (a) Triple junction in the case of perfect mirror symmetry. (b)Triple junction disappear when corners are smoothed.

The good news is that, triple junctions rarely occur in disconnected skeletons. The bad news is that they are not completely impossible. Hence, whenever a triple junction is detected, both stable alternatives are kept as multiple parses.

2.2 Canonical Coordinate Frame

The center point of a shape and one of the major axes can be used to set up a canonical coordinate frame (Figure 2.4). The line connecting the shape center and a nearby point on the selected major axis defines the reference axis. In [2], the negative major axes are used as reference axis. Spatial organization of symmetry branches and their lengths are measured according to the selected coordinate frame.

The length between the termination point of a symmetry branch and the shape center is called r . The angle between the reference axis and the arrow connecting the shape center and the termination point of a symmetry branch defines θ . Note that, even under significant bendings and articulations, since

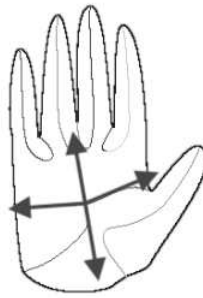


Figure 2.4: Four possible reference axes of the hand shape from [2]

the termination points of symmetry branches remain stable as demonstrated in [3, 2], they are used to determine the polar location of branches. An example coordinate frame is shown in Figure 2.5. Note that, in [2], the shape is divided into two halves for efficient matching and polar angles are calculated in counter-clockwise direction.

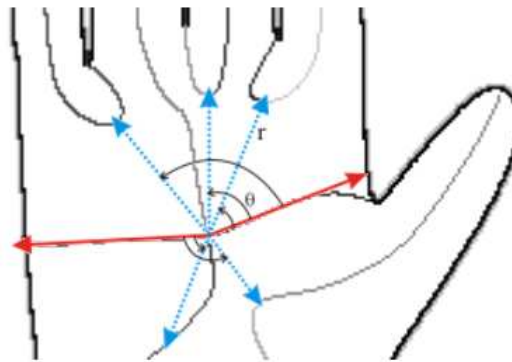


Figure 2.5: A sample coordinate frame from [2]

For each major negative branch, a different coordinate frame is created. Therefore, if there are n major negative axes that reach the shape center, n possible descriptions are generated.

CHAPTER 3

FROM DISCONNECTED SKELETON TO SHAPE TREE

In previous chapter, the computation of disconnected skeletons has been reviewed. In this chapter, ordered, unlabeled, attributed, depth-1 tree is presented as our basic data structure to represent a shape. Notice that, two important properties of the disconnected skeleton scheme in [2] that allows this study to obtain shape trees from the disconnected skeletons are: *a)* the unique shape center *b)* canonical coordinate frame.

3.1 Construction of Ordered Attributed Depth-1 Shape Tree from Symmetry Branches

We consider a disconnected skeleton as an initial shape description which will be parsed into a tree. Due to uniqueness of its center, we express a shape as a rooted tree whose nodes are the disconnected branches. Using n major negative branches, we define n rooted ordered trees by clockwise ordering to serve as alternative descriptions (parses) of the same object as illustrated in Figure 3.1. Root of the tree holds the unique shape center and the leaves hold the disconnected branches. Each leaf is associated with a set of attributes (normalized length and sign of the branches and the location of the interior (disconnection) points) presented in the previous chapter.

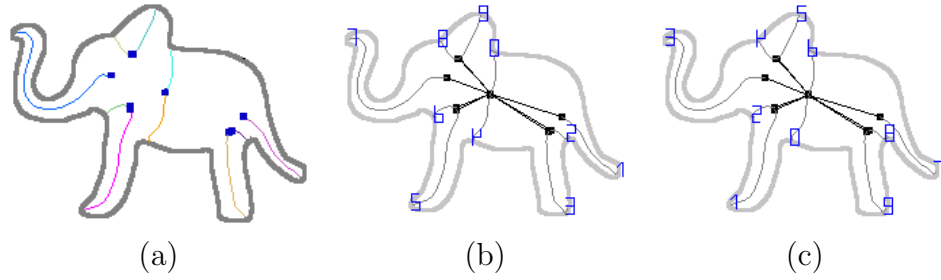


Figure 3.1: (a) Disconnected skeleton of an elephant shape. (b)-(c) Two alternative parses.


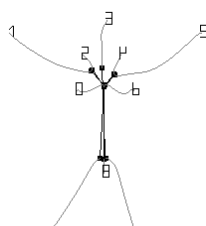
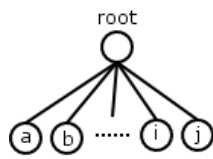
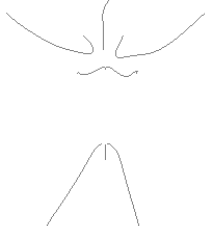
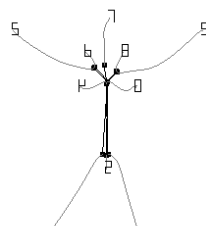
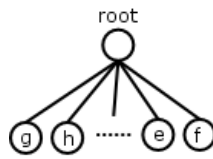

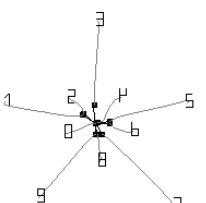
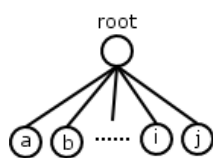
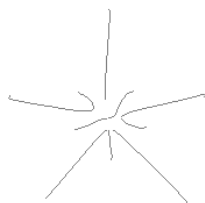
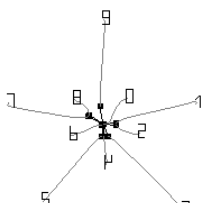
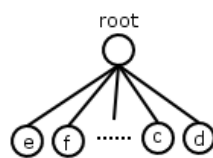
Skeleton to tree conversion process is illustrated in Table 3.1 using *human* and *star* shapes. In the first column of the table, the shapes and their disconnected skeletons are shown. Both shapes have exactly two negative branches flowing into the shape center. Therefore, two shape trees are created for each shape. The first descriptions are located in the first and third rows of the second column. The branches are numbered according to a post-ordered numbering scheme. The reference branch is numbered as 0 and other branches are numbered according to their relative order. For a branch denoting an object section, the numbering depends on the choice of reference axis. In the first description of the human shape, since the branch labeled as *a* is used as the reference branch, the node corresponding to the branch that denotes the head of the human shape is located as the fourth child of the root in the shape tree. But in the second representation of the human shape, the branch labeled as *g* is used as the reference branch. Therefore the node corresponding to the branch that denotes the head of the human shape is located as the eighth child.

Note that, the character labeling is merely used for illustrative purposes. We neither explicitly compute parts that form the shape structure nor associate any meaning to any branch.

3.1.1 Attributes of a Node in the Shape Tree

A nodal attribute can be defined as an application-dependent, user-defined value. In this study, attributes of a node are used to store the polar location, normalized length and the sign of a symmetry branch. In the canonical frame of Aslan and Tari [2], these attributes are all invariant to Euclidean transformations (scaling,

Table 3.1: Sample Skeleton to Tree Conversions

 <p>human</p>		$0 \Rightarrow a$ $6 \Rightarrow g$ $1 \Rightarrow b$ $7 \Rightarrow h$ $2 \Rightarrow c$ $8 \Rightarrow i$ $3 \Rightarrow d$ $9 \Rightarrow j$ $4 \Rightarrow e$ $5 \Rightarrow f$	 <p>first shape tree</p>
 <p>skeleton</p>		$0 \Rightarrow g$ $6 \Rightarrow c$ $1 \Rightarrow h$ $7 \Rightarrow d$ $2 \Rightarrow i$ $8 \Rightarrow e$ $3 \Rightarrow j$ $9 \Rightarrow f$ $4 \Rightarrow a$ $5 \Rightarrow b$	 <p>second shape tree</p>
 <p>star</p>		$0 \Rightarrow a$ $6 \Rightarrow g$ $1 \Rightarrow b$ $7 \Rightarrow h$ $2 \Rightarrow c$ $8 \Rightarrow i$ $3 \Rightarrow d$ $9 \Rightarrow j$ $4 \Rightarrow e$ $5 \Rightarrow f$	 <p>first shape tree</p>
 <p>skeleton</p>		$0 \Rightarrow e$ $6 \Rightarrow a$ $1 \Rightarrow f$ $7 \Rightarrow b$ $2 \Rightarrow g$ $8 \Rightarrow c$ $3 \Rightarrow h$ $9 \Rightarrow d$ $4 \Rightarrow i$ $5 \Rightarrow j$	 <p>second shape tree</p>

rotation, translation) and very insensitive to articulations and local deformations.

Polar Coordinate

Polar coordinate of end-point of a symmetry branch is used to represent the location of the branch. The location information is used to find the relative arrangement of the branches. For example, if the symmetry branch representing the head of a human shape is between the branches representing the arms, this arrangement must be preserved in representation of the shape.

Normalized Length

Normalized lengths of similar branches must also be similar. Therefore, normalized length of a branch is stored as an attribute in the corresponding node.

Sign

Because of different type of deformation, two branches with similar normalized length and polar location can not be similar, if their signs are different. Therefore, sign information of a branch must be taken into account in visual recognition processes.

3.2 Sample Shape Trees

In retrieval processes that are discussed in following chapters, shape trees are used to obtain similarity measure among shapes. Therefore, in Table 3.2, some shape trees prior to ordering are shown to give an idea about the relative arrangement of branches. Note that, the skeleton branches are also shown merely for illustrative purpose.

Table 3.2: Sample skeletons from each class in shape database

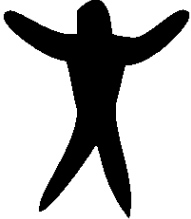
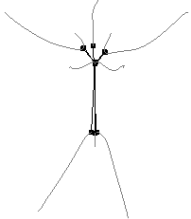

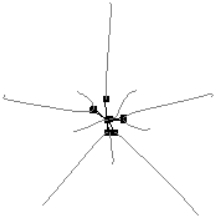

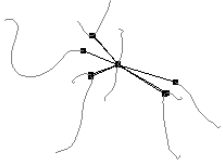

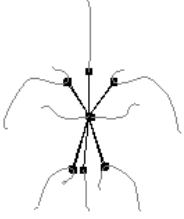

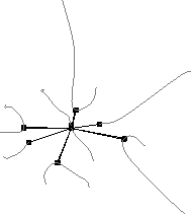

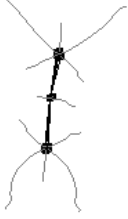
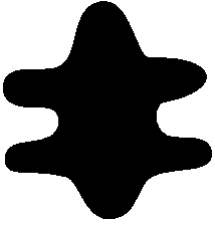
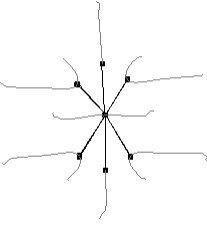

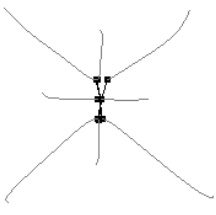

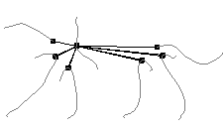

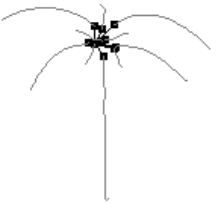

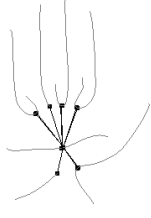


			
human		star	
			
elephant		tortoise	
			
misc1		dumbbell	
			
misc2		misc3	
			
cat		palm	
			
hand		ship	

Table 3.2 (continued)

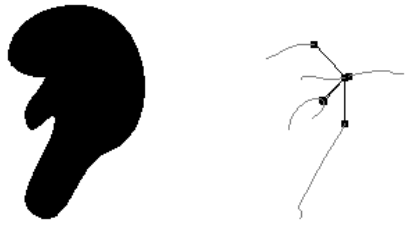

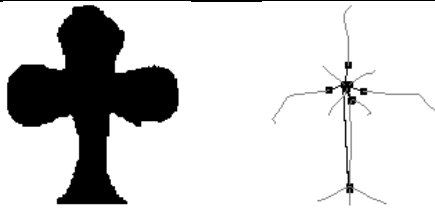
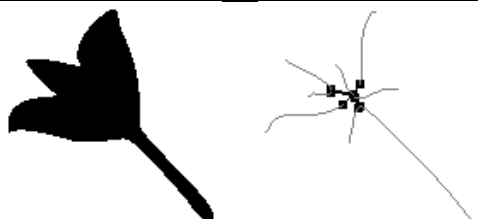
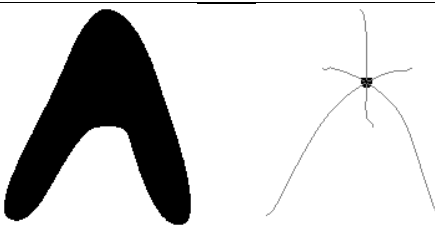

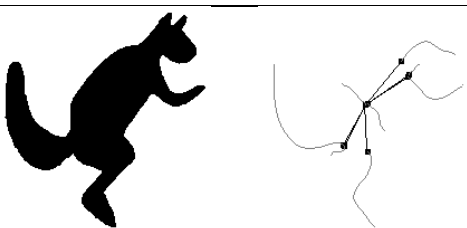
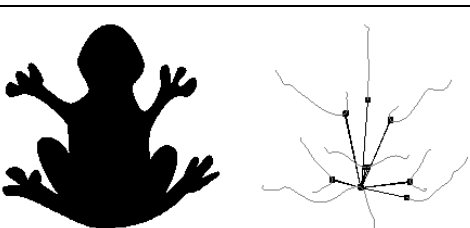
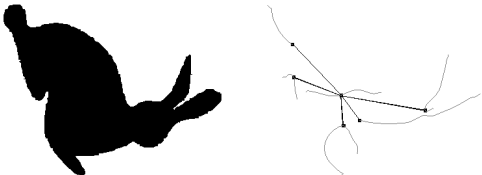
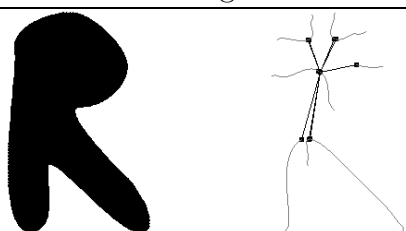
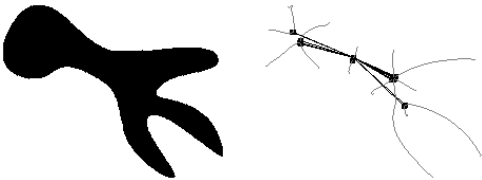
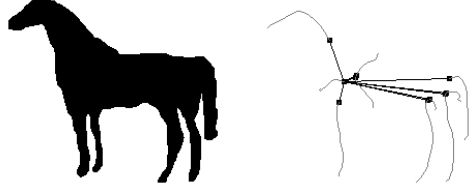

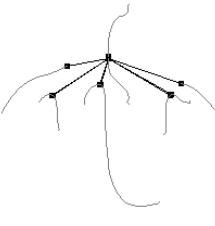

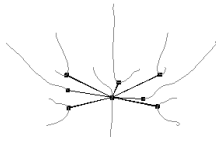

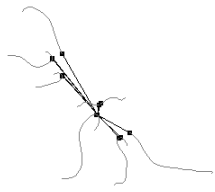

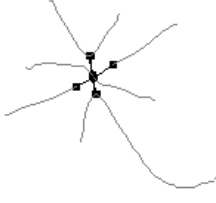

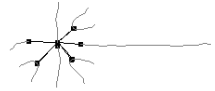

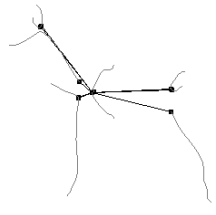
	
<p>embryo</p>	<p>turtle</p>
	
<p>club</p>	<p>flower</p>
	
<p>misc5</p>	<p>crocodile</p>
	
<p>squirrel</p>	<p>frog</p>
	
<p>rabbit</p>	<p>misc6</p>
	
<p>misc7</p>	<p>horse</p>

Table 3.2 (continued)

 	 
umbrella	crown
 	 
dino	fish
 	 
key	dog

CHAPTER 4

APPROXIMATE TREE MATCHING

In previous chapter, we demonstrated how we parse a disconnected skeleton into a tree data structure. In this chapter, we explain how we determine the best partial match between two trees, using tree editing distance.

When a shape (primitives + inclusion relations) is expressed by a tree, best correspondence or best matching between two shapes is determined by computing best partial match between the trees and the shape dissimilarity is computed as a tree editing distance.

Tree editing distance is the minimum cost of operations necessary to convert one tree to another. The edit-distance concept is originated in a paper by Wagner and Fisher [21] on comparing two character strings where they have defined three edit operations: deleting a character, inserting a character and changing one character into another.

It is quite natural to formulate both exact and approximate matches between two objects; i.e. string "*w * ing*" matches to "*widen*" approximately whereas it matches to "*willing*" and "*windsurfing*" exactly.

Most of the work on edit-distance in trees addresses the computation of distance between ordered rooted trees. Ordered trees are special kind of trees in which the left-to-right order among the siblings is important.

The comparison of rooted trees is mostly used in applications where hierarchies must be represented like parse trees and image decomposition. String

comparison is also an area where approximate tree matching is widely used. Selkow [22] presented a restricted tree-editing distance algorithm for string-like trees whose algorithmic complexity is $O(|T_1| \times |T_2|)$. The first non-exponential, non-restricted tree-editing distance algorithm had been proposed by Tai [23]. Although it could be able to solve the approximate tree matching problem, its space and time complexity is $O(|T_1| \times |T_2| \times \text{depth}(T_1)^2 \times \text{depth}(T_2)^2)$.

In [12], Zhang and Shasha proposed an algorithm with a time complexity $O(|T_1| \times |T_2| \times \min(\text{depth}(T_1), \text{leaves}(T_1)) \times \min(\text{depth}(T_2), \text{leaves}(T_2)))$. The worst case bound for this algorithm is $O(n^4)$. Klein [24] has proposed an algorithm with a better worst case bound, which is $O(n^3 \log(n))$. However, the performance of these two algorithms depends on the shape of trees that are being compared.

The focus of this chapter is to review Shasha and Zhang's ([12]) tree editing distance algorithm, as it forms the backbone of our matching scheme. Although the original algorithm that calculates the editing-cost between two trees does not create a mapping, it is possible to use back tracking on the set of whole editing operations to find the correct mapping. The algorithm that finds the mapping between two trees is presented in 4.2.

4.1 Tree Editing Distance Algorithm of Shasha and Zhang

Edit Operations

To convert one tree to another, three operations - attribute change, delete, insert- are used. The operations are illustrated in Figure 4.1. The attribute change operation is used to change the attribute of a node and has the meaning "The attached properties of a given node is changed from a to b ". When delete operation is applied to a node, all of its children becomes the children of its parent.

The last editing operation is the insert operation. In Figure 4.1(c) a sample insert operation is presented. The location of the new node is determined by the context and this issue is clarified in the algorithm section.

Generally, an editing operation is denoted as $a \Rightarrow b$. When $a \neq \Lambda$ and

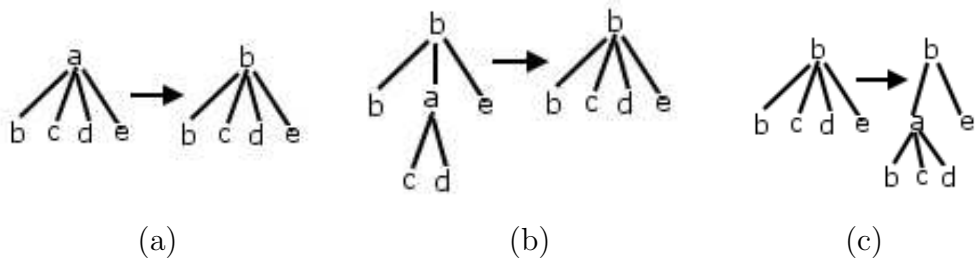


Figure 4.1: Edit operations (a)label change (b)delete (c)insert

$b \neq \Lambda$, the operation is a change operation where Λ denotes an empty node. When $a \neq \Lambda$ and $b = \Lambda$, the operation is a delete operation and when $a = \Lambda$ and $b \neq \Lambda$, the operation is an insert operation. $a \Rightarrow b$ will be used as the general editing operation in the rest of this document.

Definitions

Each editing operation needed to convert one tree to another has a cost. Cost function of editing operation $a \Rightarrow b$ is denoted as $\gamma(a \Rightarrow b)$.

Since the goal of tree editing distance algorithms is to find the minimum editing cost necessary to convert one tree to another, we will denote the tree to be modified as T_1 and the reference tree as T_2 .

As a result of editing operations, a graphical representation can be obtained that shows deleted nodes of the first tree, deleted nodes of the second tree and related nodes of the first and the second tree. In Figure 4.2, node with label b in T_1 is deleted. The lines from T_1 to T_2 show the relations between two trees. A line from T_1 to T_2 means, the node at the beginning of the line corresponds to the node that is at the end of the line. The whole relation is called a mapping.

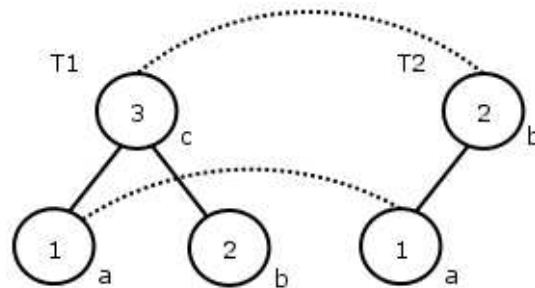


Figure 4.2: A sample mapping

Let $T[i]$ be the i^{th} node of the tree in the specified numbering scheme. In notation $T[i]$, i is the unique id number of the node.

Let $l(i)$ be the unique number of the left-most-leaf of the sub-tree rooted at $T[i]$. The concept is illustrated in Figure 4.3. Note that if $T[i]$ is a leaf, $l(i)=i$.

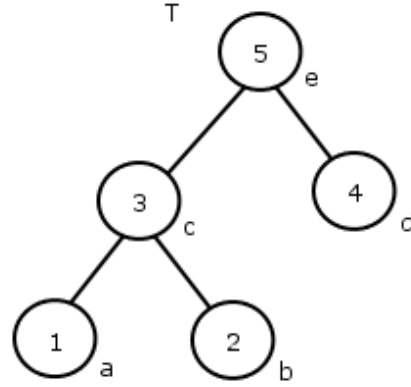


Figure 4.3: $l(5) = 1$ and $l(2) = 2$

$depth(i)$ is used to denote the depth of $T[i]$.

$T[i..j]$ is the collection of nodes from $T[i]$ to $T[j]$. This collection is called ordered sub-forest. Figure 4.4 shows $T[2..4]$ of the tree in Figure 4.3.

If $i > j$ then $T[i..j] = \emptyset$. $T[l(i)..i]$ is referred as $tree(i)$. $LR_keyroots$ of a tree is defined as :

$$LR_keyroots(T) = \{k | \text{there exists no } k' > k \text{ such that } l(k) = l(k')\}$$

This definition means k is either the root of the tree or it has no left siblings, where k denotes the unique id number of the node. Consider the tree in Figure 4.3. By definition, root of the tree, 5, is in $LR_keyroots$ list. Since $l(2)=2$ and there exists no node with a greater unique id number and same l , 2 is also in $LR_keyroots$ list. So $LR_keyroots$ of the tree in Figure 4.4 are $\{2, 4, 5\}$.

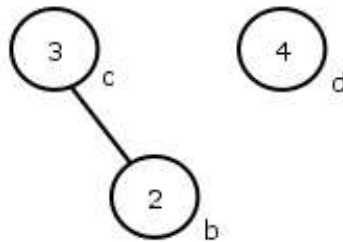


Figure 4.4: $T[2..4]$ of the tree in Figure 4.3

Editing distance between sub-forests $T_1[i'..i]$ and $T_2[j'..j]$ is denoted as $forestdist(T_1[i'..i], T_2[j'..j])$ or $forestdist(i'..i, j'..j)$.

The Algorithm

In this section, the tree editing algorithm in [12] and mapping algorithm that has been used in this study are presented. The tree editing algorithm is presented to create a notion about the usage of editing operations, since these operations are modified to compare two shapes in this study. Details of the algorithm and proofs can be found in the original paper.

The following pseudo-code presents the main loop of the algorithm. Note that, as a preprocessing, $LR_keyroots$ and $l()$ values of compared trees are calculated.

Algorithm *FindDistance()*

Input: Tree T_1 and T_2

Output: Editing distance between T_1 and T_2

(* Compute $l()$ for T_1 and T_2 , $LR_keyroots(T_1), LR_keyroots(T_2)$ as preprocessing *)

1. **for** $i' \leftarrow 1$ **to** $|LR_keyroots(T_1)|$
2. **for** $j' \leftarrow 1$ **to** $|LR_keyroots(T_2)|$
3. $i = LR_keyroots[i']$
4. $j = LR_keyroots[j']$
5. compute $treedist(i, j)$
- 6.

Dynamic programming is used to compute $treedist(i, j)$. So a computed value does not need to be computed again. The following pseudo code presents the algorithm that computes $treedist(i, j)$.

Algorithm *treedist(i, j)*

1. $forestdist(\phi, \phi) = 0$
2. **for** $i_1 \leftarrow l(i)$ **to** i
3. $forestdist(T_1[l(i)..i_1], \phi) = forestdist(T_1[l(i)..i_1 - 1], \phi) + \gamma(T_1[i_1] \Rightarrow \Lambda)$
4. **for** $j_1 \leftarrow l(j)$ **to** j
5. $forestdist(\phi, T_2[l(j)..j_1]) = forestdist(\phi, T_2[l(j)..j_1 - 1]) + \gamma(\Lambda \Rightarrow T_2[j_1])$
6. **for** $i_1 \leftarrow l(i)$ **to** i

```

7.         for  $j_1 \leftarrow l(j)$  to  $j$ 
8.             if  $l(i_1) = l(i)$  and  $l(j_1) = l(j)$ 
9.                  $forestdist(T_1[l(i)..i_1], T_2[l(j)..j_1]) = \min\{$ 
10.                     $forestdist(T_1[l(i)..i_1 - 1], T_2[l(j)..j_1]) + \gamma(T_1[i_1] \Rightarrow \Lambda),$ 
11.                     $forestdist(T_1[l(i)..i_1], T_2[l(j)..j_1 - 1]) + \gamma(\Lambda \Rightarrow T_2[j_1]),$ 
12.                     $forestdist(T_1[l(i)..i_1 - 1], T_2[l(j)..j_1 - 1]) + \gamma(T_1[i_1] \Rightarrow T_2[j_1])\}$ 
13.
14. (* put in permanent array *)
15.          $treedist(i_1, j_1) = forestdist(T_1[l(i)..i_1], T_2[l(j)..j_1])$ 
16.     else
17.          $forestdist(T_1[l(i)..i_1], T_2[l(j)..j_1]) = \min\{$ 
18.             $forestdist(T_1[l(i)..i_1 - 1], T_2[l(j)..j_1]) + \gamma(T_1[i_1] \Rightarrow \Lambda),$ 
19.             $forestdist(T_1[l(i)..i_1], T_2[l(j)..j_1 - 1]) + \gamma(\Lambda \Rightarrow T_2[j_1]),$ 
20.             $forestdist(T_1[l(i)..i_1 - 1], T_2[l(j)..j_1 - 1]) + treedist(i_1, j_1)\}$ 

```

The time complexity of the algorithm is $O(|T_1| \times |T_2| \times \min(\text{depth}(T_1), \text{leaves}(T_1)) \times \min(\text{depth}(T_2), \text{leaves}(T_2)))$ and the space complexity is $O(|T_1| \times |T_2|)$. The proofs can be found in [12].

4.2 Mapping Algorithm

In this section, the used back tracking algorithm for mapping is discussed. Note that a left-to-right post-order numbering scheme is used for tree nodes.

The editing distance between two trees is calculated from a set of sub-forest distance costs. One can find the correct mapping between the nodes of compared trees by traversing the sub-forest costs that are used in editing distance cost calculation. For each forest-distance calculation, two sub-forests and an operation are stored in a list to find the sub-forest that has been used in the calculation of forest-distance. For example, for the forest-distance:

$$forestdist(i_1..i_2, j_1..j_2) = forestdist(i_1..i_3, j_1..j_2) + \gamma(T_1[i_2]) \rightarrow T_2[i_3])$$

$forestdist(i_1..i_2, j_1..j_2)$ is stored as resultant forest-distance, $forestdist(i_1..i_3, j_1..j_2)$ is stored as used forest-distance and $ATTRIBUTE_CHANGE$ is stored as the used operation.

Before going into the details of the algorithm, some definitions have to be done. Let *editOps* be the list that contains the set of whole editing operations. Every element in the list contains a resultant forest-distance, used forest-distance and a used operation. For the i^{th} element of the list, these values are denoted as, *editOps*[i].*resForestDist*, *editOps*[i].*usedForestDist* and *editOps*[i].*usedOp* respectively. Let *results* be the stack that stores the mapping. A node is pushed into the stack with its operation. For example, for the compared trees T_1 and T_2 , if the i^{th} node of T_1 is detected as deleted, it is pushed into the stack as *results.push*(i , *deleted*). The elements of the stack gives the correct mapping with inserted and deleted nodes.

The following pseudo-code fills the *results* stack. Note that, the list *editOps* had to be filled with the whole set of forest-distance calculation operations, before calling the following function.

Algorithm *FindMapping*(*index*, *editOps*, *results*)

Input: Index of the list element, list of editing operations, stack that holds results

1. $subforest(i_1^1..i_2^1, j_1^1..j_2^1) = editOps[index].resForestDist$
2. $subforest(i_1^2..i_2^2, j_1^2..j_2^2) = editOps[index].usedForestDist$
3. **if** *editOps*[*index*].*usedOp* = *DELETE*
4. **for** $i \leftarrow (i_2^2 + 1)$ **to** i_2^1
5. **if** $i \neq 0$ *results.push*(i , *deleted*)
6. **if** *editOps*[*index*].*usedOp* = *INSERT*
7. **for** $i \leftarrow (j_2^2 + 1)$ **to** j_2^1
8. **if** $i \neq 0$ *results.push*(i , *inserted*)
9. **if** *editOps*[*index*].*usedOp* = *ATTRIBUTE_CHANGE*
10. **if** $(i_2^1 - i_2^2) \geq (j_2^1 - j_2^2)$
11. $s \leftarrow i_2^1, k \leftarrow j_2^1, l \leftarrow j_2^2, m \leftarrow i_2^2 + 1$
12. **else**
13. $s \leftarrow j_2^1, k \leftarrow i_2^1, l \leftarrow i_2^2, m \leftarrow j_2^2 + 1$
14. **for** $i \leftarrow k$ **to** l $i --$
15. *results.push*(s , i)
16. $s --$
17. **for** $i \leftarrow m$ **to** s
18. **if** $(i_2^1 - i_2^2) \geq (j_2^1 - j_2^2)$

```

19.             results.push(i, deleted)
20.         else
21.             results.push(i, inserted)
22. if ( $j_1^2 = 0$ )and( $j_2^2 = 0$ )
23.     for  $i \leftarrow i_1^2$  to  $i_2^2$ 
24.         if  $i \neq 0$  results.push(i, deleted)
25.     return
26. if ( $i_1^2 = 0$ )and( $i_2^2 = 0$ )
27.     for  $i \leftarrow j_1^2$  to  $j_2^2$ 
28.         if  $i \neq 0$  results.push(i, inserted)
29.     return
30. find index of subforest( $i_1^2..i_2^2, j_1^2..j_2^2$ ) in editOps list and assign it to newIndex
31. FindMapping[newIndex, editOps, results]

```

CHAPTER 5

SHAPE MATCHING USING TREE EDITING DISTANCE

In this chapter, we utilize the editing distance algorithm given in Chapter 4 to compute the best partial correspondence between two shape trees and the corresponding cost of morphing one shape tree into another. The key idea in these computations is using the shape primitives in editing cost calculations. Because of insertion and deletion costs, not only the scalar values of primitives but also their spatial organization play role on calculation of similarity. Note that, in [2], two shapes are compared with a branch and bound algorithm that uses the same shape primitives. By using a tree comparison algorithm, the algorithmic complexity of the matching operation is reduced.

If two trees are identical, the cost of converting one tree to the other is 0 and the cost increases proportional to the degree of dissimilarity between the trees. Therefore, the conversion cost is a measure of dissimilarity. Since dissimilarity is reversely proportional to similarity, a tree editing cost can be used as a similarity measure.

To test the approach, shape retrieval problem is used. The experiments are performed on the shape database shown in Figure 1.2. Note that, the correspondence between two shape trees is symmetric.

5.1 Calculation of Dissimilarity Measure

The idea behind the computation of the measure of dissimilarity between two shapes is calculating the tree editing distance between all descriptions of the shapes and using the minimum cost as the measure of dissimilarity. To use the tree editing distance algorithm in [12] for shape trees, shape primitives must be incorporated in the calculation of tree-to-tree conversion cost. By using shape primitives in editing operations, the tree-to-tree conversion cost becomes the dissimilarity measure between compared shape trees. The focus of this section is to present how shape primitives can be used in editing operations.

5.1.1 Insertion and Deletion Costs

The longest branch in a shape is one of the positive main axes. Usually, the contribution of a long axis to represent the characteristics of a shape is higher than a short axis with the same sign. The importance of a long branch in a shape can easily be seen in Figure 5.1. If the longest branch is deleted as shown in Figure 5.1 (b), the remaining branches may not represent a shape similar to the original one.

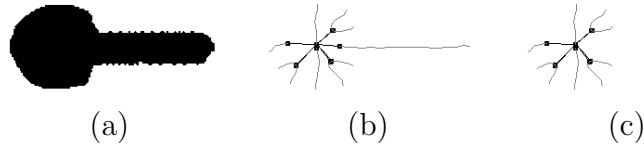


Figure 5.1: Deleting a long branch may convert the shape to a totally different shape. (a)Original shape (b)Shape tree of *a* (c) The longest branch in *b* is deleted

In calculation of deletion cost, the following rules are applied:

- the deletion cost of a branch must decrease, as the distance between the branch and the shape center increases.
- the deletion cost of a long branch must be higher than the deletion cost of a shorter branch

According to these rules, (a) the deletion cost of two branches with same length and same polar distance to shape center must be equal, (b) the deletion

cost of the longest branch must be high and deletion cost of other branches must be calculated proportional to this node.

Let (r_i^1, θ_i^1) , (r_j^2, θ_j^2) be the polar coordinates of two branch nodes i and j of the shape trees T_1 and T_2 respectively. Also let l_i^1 and l_j^2 be the normalized lengths of corresponding branches. The deletion cost is defined as:

$$\text{DELETION_COST} = (1 - r_i^1) \times \frac{l_i^1 \times \text{DEL_COST}}{l_{max}^1}$$

where l_{max}^1 is the normalized length of the longest branch of T_1 and DEL_COST is a constant which corresponds to the deletion cost of the longest branch.

Similar arguments can be made for insertion cost. If trees T_1 and T_2 are being compared, in an insertion to T_1 , the node to be inserted must already exist in T_2 . Hence, the insertion cost of the node must be calculated relative to the longest branch of T_2 . We define the insertion cost as:

$$\text{INSERTION_COST} = (1 - r_j^2) \times \frac{l_j^2 \times \text{INS_COST}}{l_{max}^2}$$

where l_{max}^2 is the normalized length of the longest branch of T_2 and INS_COST is a constant which corresponds to the insertion cost of longest branch.

5.1.2 Attribute Change Cost

If the attribute change cost is used as 0, deletion cost is used as 1 and insertion cost is used as 1, the tree editing distance algorithm in [12] can be used to compare two trees according to the relative arrangement of their nodes. In Figure 5.2, the shape trees of a human shape and a star shape are shown. If the tree editing distance algorithm is used to find the tree-to-tree conversion cost as defined above, since the number of nodes in Figure 5.2(a) is equal to the number of nodes in Figure 5.2(b), the cost is calculated as 0. Therefore, attributes of nodes must incorporate in the calculation of attribute change cost.

For the calculation of attribute change cost, the following rules are applied:

- if two branches are similar, their lengths must be similar
- if two branches are similar, their polar coordinates must be similar

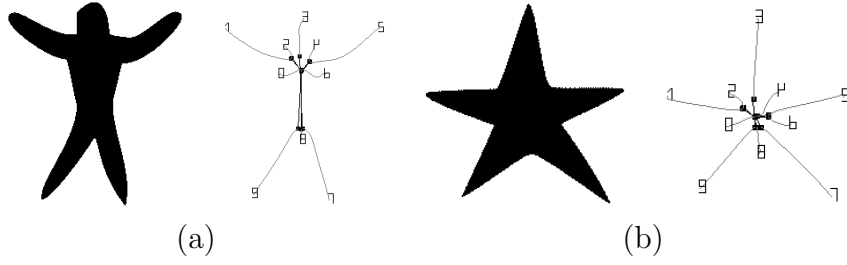


Figure 5.2: Number of nodes in shape trees of different shapes may be equal

According to these rules, if two branches are identical, the attribute change cost between the nodes corresponding to these branches must be 0 and the cost increases as the similarity between branches decreases. The attribute change cost (*ATTRIBUTE_CHANGE*) is defined as:

$$2 \times \frac{\frac{\max(l_i^1, l_j^2) - \min(l_i^1, l_j^2)}{\max(l_i^1, l_j^2)} + \frac{\max(r_i^1, r_j^2) - \min(r_i^1, r_j^2)}{\max(r_i^1, r_j^2)} + \frac{\max(\theta_i^1, \theta_j^2) - \min(\theta_i^1, \theta_j^2)}{\max(\theta_i^1, \theta_j^2)}}{4}$$

where (r_i^1, θ_i^1) , (r_j^2, θ_j^2) are the polar coordinates of two branch nodes i and j of the shape trees T_1 and T_2 respectively and l_i^1 and l_j^2 are the normalized lengths of corresponding branches.

For the attribute change cost, more emphasize is given to normalized length of branches. As will be discussed later, this behavior is not always valid and may cause mismatches for some shapes.

5.2 Mapping

The editing operations give rise to a mapping that is a graphical specification of matching branches of shape trees. In Figure 5.3, two sample mapping results are shown. In Figure 5.3(a), the branches of two shapes that are in the same category are shown. Despite significant articulations, correct correspondence between the branches are found. In Figure 5.3(b), the mapping of branches of two shapes from different categories is presented. Since not only the branch lengths, but also the polar locations of branches are used in tree editing distance calculations, similar parts of objects are mapped.

The mapping process is also successful for objects with missing parts. In Figure 5.4, the mapping result of a normal hand shape and a hand shape with

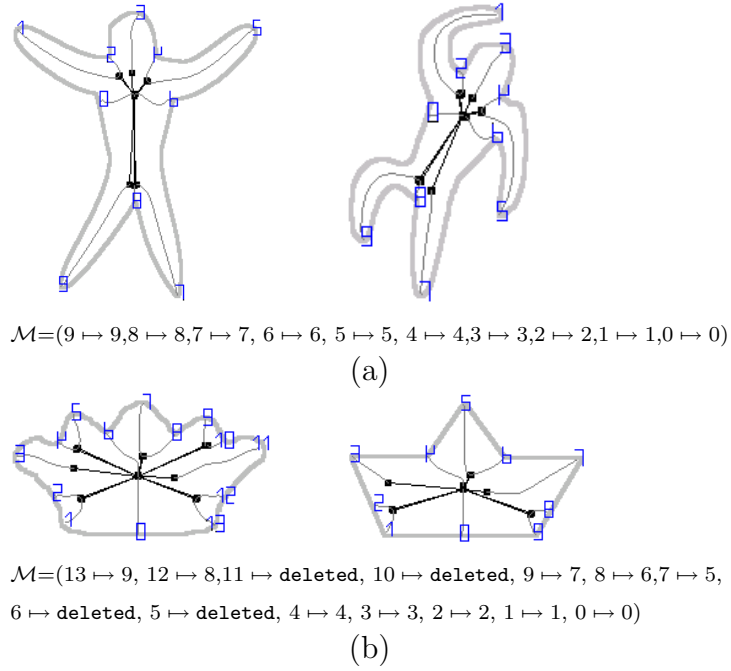


Figure 5.3: Sample mappings

a missing finger is shown. The branches that correspond to the missing finger (the branches that are numbered as 6 and 7) are deleted and other branches of two shapes are matched properly.

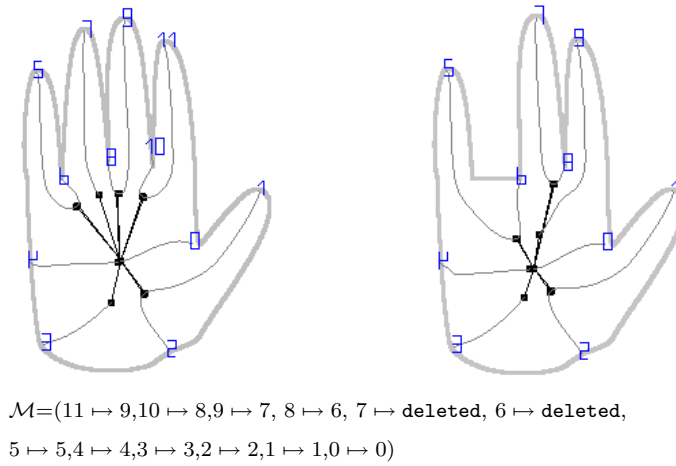


Figure 5.4: Mapping of two hand shapes one with a missing finger

5.3 Experiments

After computing all costs for all shape pairs in the database (total 180×180), each one of the 180 shapes is selected as a query shape (stimulus input) and the

top twelve (twice the number of shapes in a category) lowest cost matches are retrieved. The retrieval rate is 98.7%. This scoring, referred to as Bull’s eye test, is the common practice [25, 10]. If we consider only the top 6 matches, retrieval rate is 92.3%. The results are presented in Table 5.1. Note that, number of wrong retrievals in the top 5 category are [1,6,13,21,41] respectively.

5.3.1 Clustering Results

Normalized cuts clustering algorithm [26] is a graph based clustering algorithm that is used for perceptual grouping of a feature space where the nodes of the graph are the points in this space and an edge is formed between every node whose weight is a function of similarity between corresponding nodes. After applying the matching algorithm to the image database, we obtain a symmetric matrix that contains the degree of dissimilarity between each shape. If we convert this dissimilarity matrix into a similarity matrix, we can apply normalized cuts clustering to obtain an idea about the perceptual grouping of the shapes.

To obtain a similarity matrix, the following function has been applied to the dissimilarity measures between shapes.

$$f(x) = \frac{1}{e^{\sqrt{x}}} \quad (5.1)$$

In Figure 5.5, a subset of the database which is used in clustering experiments is shown. When these shapes are grouped into three disjoint sets by using normalized cuts algorithm, three clusters shown in Figure 5.6 are obtained. As seen in Figure 5.6, the dumble-like shapes form one group, star-like shapes form one group and remaining shapes form another group.

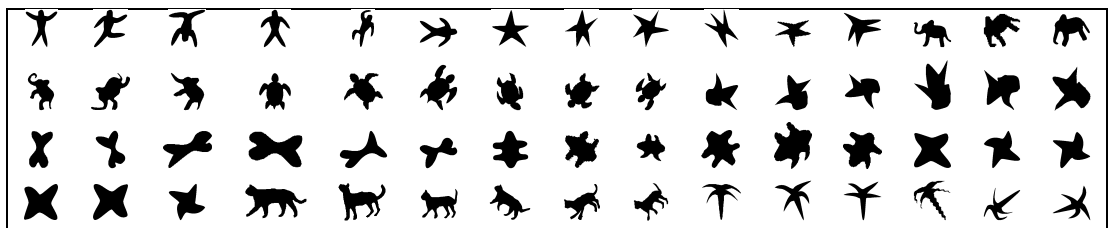


Figure 5.5: Sample shapes to be clustered with NCuts clustering algorithm

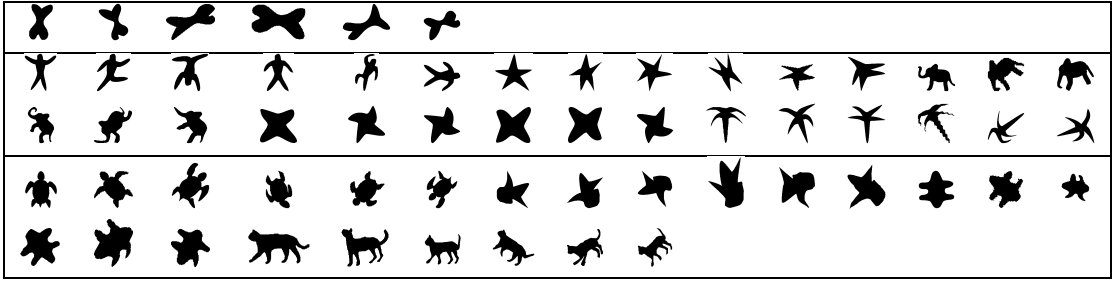


Figure 5.6: Shapes in 5.5 clustered into 3

When the shapes shown in 5.5 are separated into 5 groups, the clusters shown in Figure 5.7 are obtained. In this case, tortoise-like shapes form the first group, four leg animals form the second group, star-like shapes form the third group, dumble-like shapes form the fourth group and remaining shapes which are also in the same class form the last group.

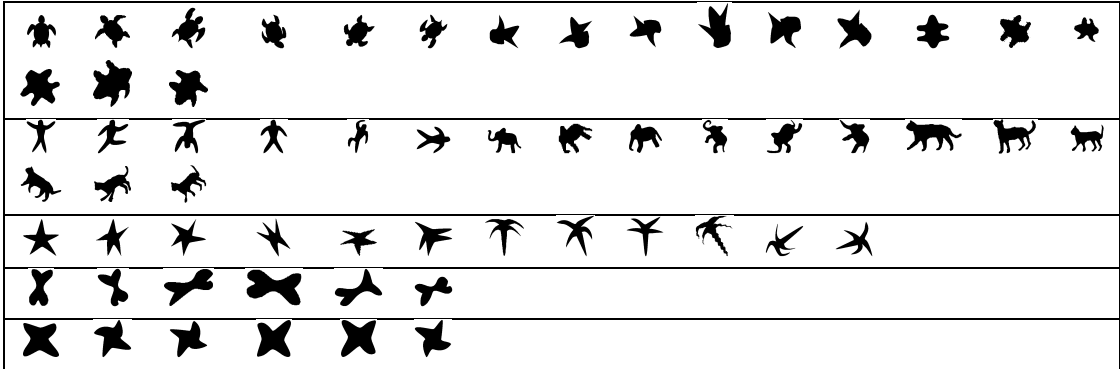


Figure 5.7: Shapes in 5.5 clustered into 5

When the shapes shown in 5.5 are separated into 10, the clusters shown in Figure 5.8 are obtained. As presented in the results, shapes in classes cat, misc1, cat, misc3, elephant and human are perceptually grouped perfectly. Although shapes from two different classes form the first and third groups, the shapes are structurally similar. Also, the high dissimilarity measure among dumbbell shapes has split the class into 3 more similar subclasses.

5.4 Discussions

The aim of this section is to discuss the behaviors that causes mismatches, to give a notion about how matching can be improved. We categorize the sources of mismatches into 4 and analyze them separately.

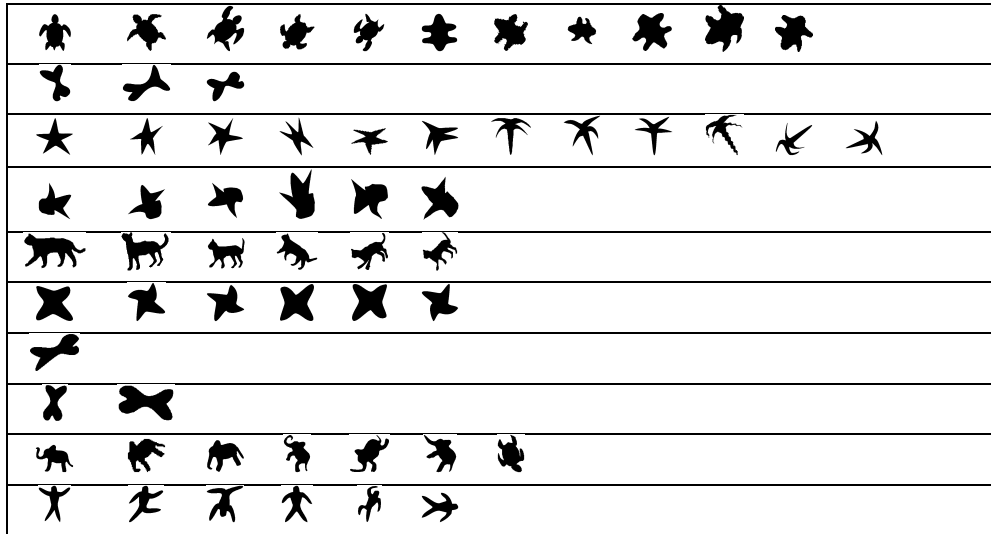
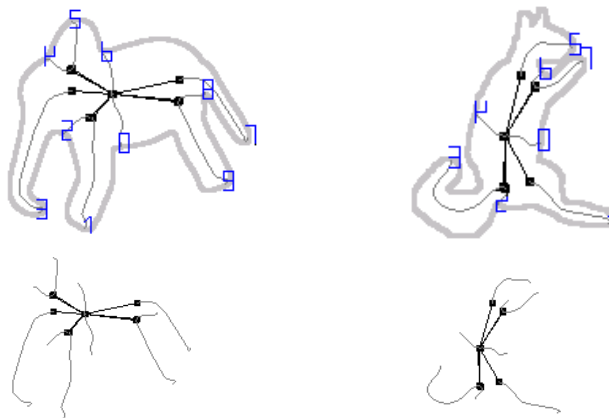


Figure 5.8: Shapes in 5.5 clustered into 10

5.4.1 Structurally Different Shapes with Similar Shape Trees

Two perceptually dissimilar shapes may have similar shape trees. A sample situation is illustrated in Figure 5.9. One can observe the close resemblance between shape trees of elephant and squirrel when branches 4 and 5 are deleted from the shape tree of the former. Since branches 4 and 5 of the elephant are short branches, their deletion cost is small and this small penalty gives rise to a wrong match. Clearly, assigning importance by branch length is not always valid. Branches 4 and 5 are two characteristic branches of an elephant shape. Therefore, they should not be treated as spurious branches and the penalty of deletion must be high to overcome this kind of mismatches.

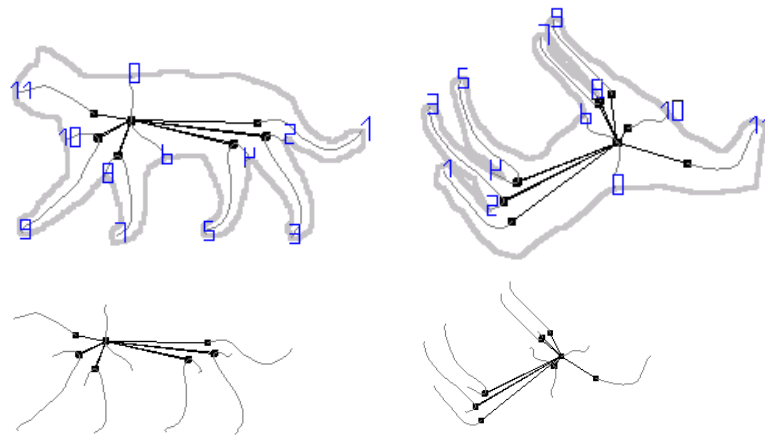


$$\mathcal{M}=(9 \mapsto 7,8 \mapsto 6,7 \mapsto 5, 6 \mapsto 4, 5 \mapsto \text{deleted}, 4 \mapsto \text{deleted}, 3 \mapsto 3,2 \mapsto 2,1 \mapsto 1,0 \mapsto 0)$$

Figure 5.9: Similar skeletons for an elephant shape and a squirrel shape

5.4.2 Structurally Similar Shapes

The second type of problematic shapes contains shapes that look like each other. Perceptually similar shapes have naturally similar shape trees. Notice that, in Figure 5.10, the cat shape and horse shape have same number of branches with similar characteristics. Therefore, the dissimilarity measure between these shapes is similar to the dissimilarity measures among cat shapes.



$$\mathcal{M}=(11 \mapsto 11, 10 \mapsto 10, 9 \mapsto 9, 8 \mapsto 8, 7 \mapsto 7, 6 \mapsto 6, 5 \mapsto 5, 4 \mapsto 4, 3 \mapsto 3, 2 \mapsto 2, 1 \mapsto 1, 0 \mapsto 0)$$

Figure 5.10: Similar shapes. A cat shape and a horse shape

5.4.3 Shapes with Extra Branches

Indentations and protrusions may give rise to extra branches in a shape. During the matching process, these extra features increase the matching cost and may cause mismatches. In Figure 5.11, two embryo shapes are shown one with extra branches. The extra branches in the second embryo shape cause a high dissimilarity measure which is the source of a mismatch.



$$\mathcal{M}=(5 \mapsto 9, 4 \mapsto 8, 3 \mapsto 7, \textit{inserted} \mapsto 6, \textit{inserted} \mapsto 5, 2 \mapsto 4, \textit{inserted} \mapsto 3, \textit{inserted} \mapsto 2, 1 \mapsto 1, 0 \mapsto 0)$$

Figure 5.11: Problematic embryo shapes one with more branches

5.4.4 Outliers in Distribution of Lengths of Branches

Certain irregularities in the distribution of branch lengths may also cause problems, i.e the existence of an outlier branch. Since the insertion/deletion costs are calculated relative to the longest branch, an outlier causes short branches to have small insertion/deletion costs. An example shape is shown in Figure 5.12 (a). Most of the branches are short and there exists a long branch which reduces the insertion/deletion costs of other branches although the sort branches are also important. Since deleting a short node is not too costly, the short branches in Figure 5.12 (a) are deleted to convert the shape tree into the shape tree in Figure 5.12 (b) and causes a mismatch. As also became clear in 5.4.1 and 5.4.3, using branch length as a significance measure is not always valid.

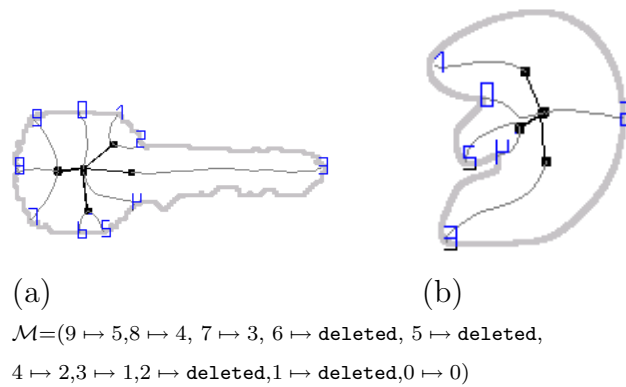


Figure 5.12: A mismatch because of an outlier. (a)key shape (b)misc5 shape

Table 5.1: Retrieval results






























































































































































































































												
	0.622	0.683	0.704	0.748	0.774	1.808	1.887	1.894	1.989	2.020	2.060	
												
	0.553	0.622	0.711	0.841	0.885	1.857	1.900	1.903	1.931	1.974	1.992	
												
	0.487	0.553	0.774	0.802	0.907	1.802	1.931	1.954	1.960	1.998	2.011	
												
	0.291	0.748	0.752	0.802	0.841	1.785	1.792	1.799	1.930	1.940	1.968	
												
	0.487	0.683	0.711	0.752	0.806	1.857	1.931	1.998	2.022	2.030	2.035	
												
	0.291	0.704	0.806	0.885	0.907	1.734	1.840	1.877	1.878	1.942	1.958	
												
	0.904	1.245	1.341	1.460	1.510	1.610	1.711	1.748	1.817	2.034	2.128	
												
	0.904	1.224	1.305	1.337	1.378	1.396	1.680	1.728	1.995	2.002	2.073	
												
	0.864	1.109	1.378	1.402	1.510	1.658	1.741	1.758	1.806	1.914	1.971	
												
	0.795	1.023	1.396	1.402	1.460	1.661	1.734	1.785	1.785	1.802	1.808	
												
	0.795	0.821	1.109	1.305	1.341	1.812	1.821	1.859	1.942	1.968	2.011	
												
	0.821	0.864	1.023	1.224	1.245	1.745	1.773	1.776	1.783	1.816	1.840	
												
	0.906	0.994	1.040	1.157	1.459	1.492	1.536	1.632	1.646	1.734	1.734	
												
	0.846	0.994	1.033	1.420	1.580	1.676	1.691	1.695	1.715	1.744	1.757	
												
	0.901	0.906	1.033	1.122	1.657	1.663	1.842	1.848	1.848	1.909	1.946	
												
	1.122	1.157	1.420	1.536	1.562	1.772	1.782	1.798	1.825	1.899	1.899	
												
	1.459	1.562	1.663	1.708	1.744	1.958	1.963	1.967	2.032	2.158	2.158	

Table 5.1 (continued)

	0.846	0.901	1.040	1.536	1.646	1.708	1.733	1.743	1.779	1.835	1.835
	1.124	1.300	1.402	1.412	1.542	1.676	1.699	1.749	2.004	2.028	2.100
	0.900	1.124	1.440	1.443	1.490	1.580	1.601	1.808	1.941	1.994	2.123
	0.542	1.290	1.443	1.503	1.542	1.544	1.590	1.590	1.709	2.116	2.284
	1.490	1.599	1.676	1.715	1.914	1.944	1.950	1.989	2.040	2.065	2.107
	0.900	1.300	1.338	1.424	1.425	1.590	1.599	1.718	1.784	1.815	2.012
	0.542	1.370	1.483	1.522	1.580	1.599	1.671	1.749	1.784	2.232	2.550
	0.628	1.488	1.644	1.749	1.992	2.341	2.395	2.426	2.429	2.430	2.436
	0.628	1.649	1.655	1.770	1.931	2.275	2.325	2.394	2.427	2.448	2.484
	1.488	1.649	1.667	1.681	1.985	2.286	2.312	2.319	2.384	2.385	2.458
	1.555	1.681	1.749	1.770	1.907	2.185	2.352	2.374	2.394	2.439	2.463
	1.624	1.644	1.655	1.667	1.907	2.526	2.535	2.546	2.599	2.628	2.628
	1.555	1.624	1.931	1.948	1.970	1.985	1.992	2.076	2.131	2.148	2.199
	0.572	1.202	1.359	1.430	1.514	2.599	2.692	2.770	3.003	3.251	3.279
	0.892	1.286	1.436	1.514	1.960	2.245	2.626	2.747	2.922	3.005	3.380
	1.359	1.432	1.591	1.914	1.960	2.902	3.054	3.110	3.127	3.282	3.327
	0.572	1.337	1.432	1.436	1.483	2.730	2.858	3.130	3.316	3.320	3.501

Table 5.1 (continued)

	1.202	1.205	1.286	1.337	1.591	2.559	2.606	2.750	2.932	3.057	3.090
	0.892	1.205	1.430	1.483	1.914	2.269	2.513	2.650	2.929	2.998	3.319
	0.595	0.980	1.219	1.290	1.370	1.412	1.424	1.440	1.444	1.455	1.914
	0.782	0.897	0.980	1.078	1.338	1.483	1.503	1.699	1.759	1.808	2.402
	0.595	0.782	1.187	1.312	1.359	1.402	1.425	1.544	1.601	1.671	1.950
	1.312	1.444	1.706	1.759	1.848	1.989	2.012	2.100	2.116	2.169	2.232
	0.897	0.967	1.359	1.455	1.522	1.590	1.815	1.848	1.941	2.028	2.572
	0.967	1.078	1.187	1.219	1.599	1.706	1.709	1.718	1.994	2.004	2.436
	0.166	0.262	0.692	0.710	0.717	1.477	1.479	1.496	1.517	1.518	1.567
	0.084	0.166	0.618	0.641	0.710	1.136	1.495	1.519	1.553	1.600	1.604
	0.084	0.171	0.603	0.647	0.692	1.156	1.488	1.495	1.540	1.580	1.582
	0.166	0.189	0.603	0.618	0.649	1.513	1.521	1.559	1.572	1.612	1.686
	0.189	0.262	0.641	0.644	0.647	1.502	1.525	1.571	1.582	1.613	1.673
	0.166	0.171	0.644	0.649	0.717	1.171	1.432	1.529	1.561	1.608	1.614
	0.757	1.256	1.466	1.483	1.487	1.941	1.989	2.085	2.186	2.243	2.252
	1.031	1.116	1.461	1.487	1.527	1.731	1.920	2.002	2.069	2.156	2.329
	1.033	1.116	1.466	1.474	1.531	1.723	1.764	1.781	1.993	2.017	2.060

Table 5.1 (continued)

	0.757	1.175	1.256	1.527	1.531	1.970	2.028	2.046	2.163	2.171	2.177	
	1.256	1.461	1.474	1.483	1.555	1.606	1.646	1.840	1.906	2.065	2.265	
	1.031	1.033	1.175	1.256	1.518	1.555	1.827	1.908	1.930	1.949	1.958	
	0.994	1.218	1.531	1.536	1.566	1.971	2.034	2.068	2.081	2.287	2.352	
	0.374	1.146	1.220	1.372	1.536	1.680	1.745	1.806	1.817	1.821	1.842	
	0.994	1.129	1.224	1.372	1.445	1.658	1.711	1.816	2.016	2.068	2.073	
	1.028	1.181	1.220	1.224	1.531	1.758	1.867	1.908	2.081	2.224	2.238	
	1.028	1.129	1.146	1.218	1.237	1.337	1.610	1.661	1.741	1.783	1.859	
	0.374	1.181	1.237	1.445	1.566	1.728	1.748	1.773	1.785	1.812	1.914	
	0.914	1.085	1.095	1.501	1.890	2.940	2.991	3.072	3.123	3.139	3.157	
	0.914	1.098	1.106	1.702	1.890	2.741	2.864	3.023	3.106	3.140	3.142	
	0.264	1.095	1.098	1.568	1.661	2.551	2.795	2.930	2.964	3.000	3.014	
	1.501	1.568	1.596	1.648	1.702	2.711	2.716	2.833	2.847	2.930	2.975	
	0.264	1.085	1.106	1.596	1.671	2.505	2.804	2.894	2.935	2.935	2.988	
	1.648	1.661	1.671	1.890	1.890	2.627	2.628	2.652	2.727	2.758	2.861	
	0.558	0.783	1.029	1.048	1.049	1.643	1.763	1.814	1.835	2.056	2.066	
	0.711	0.783	1.185	1.233	1.342	1.887	1.901	1.965	2.008	2.206	2.272	

Table 5.1 (continued)

	0.558	0.711	1.178	1.200	1.226	1.844	1.882	1.912	1.974	2.136	2.243	
	0.379	0.493	1.029	1.185	1.226	1.902	1.950	1.993	1.999	2.058	2.083	
	0.185	0.379	1.048	1.178	1.233	1.853	1.904	1.950	1.966	2.029	2.075	
	0.185	0.493	1.049	1.200	1.342	1.798	1.896	1.919	1.951	1.973	2.012	
	0.329	0.364	0.697	0.770	1.337	1.511	1.664	1.672	1.708	1.737	1.741	
	0.149	0.329	0.676	0.797	1.261	1.635	1.768	1.803	1.830	1.876	1.891	
	0.770	0.788	0.797	1.055	1.588	1.671	1.674	1.711	1.718	1.728	1.733	
	0.676	0.697	0.757	1.055	1.596	1.705	1.811	1.855	1.936	1.952	1.986	
	1.261	1.301	1.337	1.596	1.733	2.115	2.163	2.178	2.189	2.197	2.219	
	0.149	0.364	0.757	0.788	1.301	1.657	1.819	1.831	1.834	1.879	1.881	
	0.572	0.750	0.952	1.083	1.403	1.751	1.810	1.816	1.848	1.902	2.039	
	0.402	0.952	0.985	1.103	1.161	1.889	1.890	1.941	2.057	2.079	2.306	
	0.327	0.572	0.985	1.046	1.345	1.869	1.898	1.941	1.952	1.969	2.116	
	0.402	1.046	1.083	1.093	1.162	1.841	1.886	1.926	2.038	2.059	2.258	
	0.327	0.750	1.093	1.103	1.465	1.841	1.867	1.889	1.909	1.914	2.102	
	1.161	1.162	1.345	1.403	1.465	2.129	2.215	2.248	2.275	2.292	2.307	
	0.746	0.778	0.943	0.972	1.580	1.600	1.608	1.666	1.780	1.789	1.792	

Table 5.1 (continued)

	0.746	1.056	1.110	1.285	1.432	1.488	1.488	1.495	1.567	1.636	1.673
	0.778	0.794	0.838	1.056	1.628	1.634	1.636	1.663	1.775	1.824	1.828
	1.136	1.156	1.171	1.383	1.395	1.461	1.466	1.502	1.507	1.517	1.521
	0.838	0.972	1.118	1.285	1.690	1.691	1.704	1.710	1.794	1.841	1.943
	0.794	0.943	1.110	1.118	1.507	1.776	1.788	1.816	1.872	1.875	1.959
	1.214	1.249	1.252	1.489	1.654	1.656	1.788	1.828	1.846	1.875	1.881
	0.483	1.218	1.339	1.394	1.656	1.728	1.808	1.816	1.858	1.875	1.876
	0.354	0.701	1.249	1.295	1.394	1.395	1.409	1.636	1.674	1.704	1.775
	0.701	0.702	1.196	1.214	1.218	1.383	1.491	1.726	1.813	1.838	1.850
	0.483	1.071	1.196	1.295	1.489	1.576	1.588	1.686	1.693	1.741	1.803
	0.354	0.702	1.071	1.252	1.339	1.461	1.534	1.671	1.692	1.789	1.794
	0.487	0.570	0.637	0.668	0.733	1.836	1.902	1.945	1.949	1.971	1.984
	0.547	0.586	0.614	0.665	0.668	1.992	2.026	2.045	2.083	2.084	2.098
	0.487	0.541	0.582	0.614	0.666	1.888	2.024	2.030	2.058	2.065	2.069
	0.411	0.541	0.547	0.573	0.637	1.959	2.006	2.011	2.052	2.056	2.057
	0.411	0.570	0.582	0.586	0.734	1.946	1.949	1.957	1.989	2.057	2.060
	0.573	0.665	0.666	0.733	0.734	2.072	2.078	2.109	2.186	2.208	2.222

Table 5.1 (continued)

	0.414	1.214	1.274	1.389	1.536	1.734	1.870	1.905	1.905	1.961	2.013
	1.286	1.532	1.536	1.713	1.772	1.825	1.836	1.836	1.843	1.897	1.955
	1.286	1.313	1.389	1.555	1.572	1.680	1.681	1.754	1.801	1.801	1.852
	1.189	1.274	1.434	1.754	1.955	2.134	2.169	2.256	2.322	2.322	2.327
	0.414	1.044	1.189	1.313	1.532	1.759	1.871	1.956	1.982	1.982	2.001
	1.044	1.214	1.434	1.852	1.897	1.924	2.118	2.128	2.128	2.147	2.172
	0.469	0.681	0.720	0.947	0.947	1.555	1.632	1.676	1.743	1.825	1.825
	0.441	0.469	0.770	0.927	0.927	1.536	1.681	1.695	1.733	1.772	1.772
	0.448	0.448	0.667	0.720	0.770	1.572	1.580	1.646	1.657	1.713	1.734
	0.441	0.667	0.681	0.873	0.873	1.492	1.646	1.680	1.691	1.798	1.842
	0.000	0.448	0.873	0.927	0.947	1.734	1.757	1.801	1.835	1.836	1.848
	0.000	0.448	0.873	0.927	0.947	1.734	1.757	1.801	1.835	1.836	1.848
	1.042	1.131	1.274	1.367	1.951	2.589	2.688	2.775	2.803	2.844	2.849
	1.131	1.137	1.545	1.767	2.070	2.640	2.737	2.857	2.869	2.901	2.912
	1.042	1.306	1.323	1.767	1.774	2.757	2.787	2.845	2.868	2.891	2.912
	1.681	1.774	1.951	2.042	2.070	2.503	2.531	2.542	2.597	2.612	2.617
	1.306	1.358	1.367	1.545	1.681	2.614	2.617	2.631	2.657	2.695	2.830

Table 5.1 (continued)

	1.137	1.274	1.323	1.358	2.042	2.452	2.565	2.598	2.643	2.698	2.699
	0.506	0.653	0.729	1.312	1.708	1.829	1.879	1.891	1.918	1.952	2.045
	1.084	1.180	1.312	1.331	1.664	1.768	1.819	1.855	2.022	2.034	2.048
	0.535	0.653	0.747	1.084	1.627	1.672	1.803	1.811	1.831	1.992	2.197
	0.506	0.747	0.859	1.331	1.737	1.819	1.915	1.925	1.952	2.008	2.083
	1.627	1.634	1.819	1.829	1.895	1.986	2.007	2.034	2.034	2.129	2.163
	0.535	0.729	0.859	1.180	1.511	1.634	1.635	1.657	1.705	1.921	2.189
	0.520	0.609	0.652	0.759	1.817	2.448	2.452	2.483	2.488	2.518	2.582
	0.258	0.343	0.520	0.990	1.889	2.238	2.239	2.275	2.361	2.429	2.480
	0.759	0.990	1.045	1.105	1.447	2.315	2.387	2.409	2.412	2.412	2.426
	0.226	0.258	0.609	1.045	2.001	2.143	2.172	2.218	2.292	2.395	2.415
	1.447	1.817	1.889	2.001	2.043	2.178	2.229	2.295	2.312	2.405	2.447
	0.226	0.343	0.652	1.105	2.043	2.229	2.235	2.264	2.354	2.430	2.458
	1.634	2.444	2.447	2.613	2.743	2.922	2.932	2.998	3.003	3.110	3.268
	1.483	1.498	2.003	2.613	2.626	2.650	2.750	2.770	3.130	3.230	3.243
	1.483	2.137	2.245	2.269	2.447	2.581	2.599	2.606	2.730	3.054	3.214
	1.634	2.003	2.309	2.372	2.581	2.929	3.005	3.127	3.232	3.251	3.370

Table 5.1 (continued)

	1.498	2.137	2.293	2.372	2.513	2.559	2.692	2.743	2.747	2.858	2.902
	2.293	2.309	2.444	3.057	3.279	3.308	3.316	3.320	3.327	3.327	3.450
	1.787	1.843	1.974	2.024	2.028	2.122	2.158	2.195	2.225	2.260	2.269
	1.204	1.480	1.606	1.872	1.920	1.949	1.993	2.107	2.282	2.329	2.337
	1.204	1.442	1.518	1.535	1.646	1.723	1.731	1.970	1.989	2.083	2.085
	0.788	1.442	1.480	1.764	1.840	1.908	1.992	2.069	2.196	2.266	2.277
	0.788	1.535	1.781	1.827	1.872	1.906	2.002	2.104	2.171	2.239	2.241
	1.787	2.046	2.252	2.291	2.314	2.329	2.397	2.441	2.470	2.670	2.679
	0.294	0.312	0.516	0.822	0.849	1.751	1.841	1.841	1.869	1.889	2.213
	0.516	0.531	0.560	0.593	0.669	1.810	1.867	1.898	1.926	1.941	2.091
	0.531	0.686	0.792	0.822	0.831	1.902	1.909	1.969	2.038	2.042	2.044
	0.312	0.403	0.593	0.806	0.831	1.816	1.889	1.952	2.057	2.059	2.170
	0.294	0.403	0.560	0.792	0.815	1.848	1.886	1.890	1.914	1.941	2.118
	0.669	0.686	0.806	0.815	0.849	2.182	2.212	2.224	2.258	2.306	2.329
	0.517	2.144	2.240	2.290	2.299	2.370	2.383	2.421	2.439	2.656	2.799
	2.127	2.240	2.717	2.790	2.854	2.888	3.060	3.094	3.136	3.150	3.159
	0.517	2.117	2.127	2.283	2.287	2.383	2.388	2.391	2.494	2.738	2.768

Table 5.1 (continued)




































































































































































































































































































































											
	1.556	1.714	2.290	2.388	2.649	2.658	2.668	2.672	2.689	2.717	2.725
											
	0.430	1.714	2.287	2.299	2.329	2.486	2.531	2.598	2.599	2.610	2.741
											
	0.430	1.556	2.117	2.144	2.453	2.621	2.645	2.646	2.711	2.730	2.730
											
	0.227	0.722	1.550	2.064	2.091	2.260	2.271	2.378	2.462	2.476	2.530
											
	0.227	0.621	1.670	2.161	2.169	2.198	2.204	2.299	2.414	2.440	2.451
											
	0.546	0.944	1.900	1.998	1.998	2.036	2.052	2.057	2.064	2.066	2.088
											
	0.546	0.806	1.887	1.903	1.929	1.931	1.931	1.971	2.037	2.064	2.073
											
	0.621	0.722	1.575	2.052	2.064	2.242	2.272	2.346	2.477	2.477	2.493
											
	0.806	0.944	1.550	1.575	1.670	1.894	1.977	2.000	2.005	2.030	2.055
											
	0.136	0.246	0.959	1.163	1.292	1.496	1.559	1.571	1.582	1.604	1.614
											
	0.522	0.959	1.001	1.042	1.293	1.614	1.667	1.686	1.706	1.723	1.726
											
	0.136	0.208	1.001	1.207	1.397	1.518	1.572	1.582	1.594	1.616	1.626
											
	1.292	1.293	1.363	1.371	1.397	1.409	1.466	1.477	1.488	1.491	1.534
											
	0.522	1.163	1.207	1.228	1.363	1.804	1.852	1.908	1.930	1.933	1.943
											
	0.208	0.246	1.042	1.228	1.371	1.479	1.495	1.513	1.519	1.525	1.529
											
	0.673	0.756	1.565	1.605	1.711	1.937	2.009	2.023	2.086	2.298	2.337
											
	0.673	1.144	1.718	1.793	1.798	1.915	1.935	1.943	2.115	2.243	2.294

Table 5.1 (continued)

											
	1.740	1.824	1.831	2.058	2.086	2.243	2.256	2.266	2.287	2.316	2.327
											
	0.756	1.144	1.500	1.506	1.753	2.012	2.058	2.087	2.102	2.384	2.486
											
	0.481	1.506	1.605	1.793	1.831	2.163	2.323	2.476	2.504	2.801	2.814
											
	0.481	1.500	1.565	1.740	1.798	2.165	2.311	2.407	2.434	2.775	2.776
											
	0.286	1.106	1.189	1.740	1.745	1.814	1.844	1.864	1.901	1.945	1.949
											
	0.267	1.692	1.700	1.745	1.792	1.815	1.834	1.877	1.919	1.947	1.950
											
	0.504	1.058	1.106	1.643	1.690	1.700	1.788	1.873	1.882	1.887	1.896
											
	0.267	1.690	1.728	1.740	1.772	1.799	1.868	1.878	1.931	1.951	1.954
											
	0.504	1.163	1.189	1.692	1.728	1.763	1.798	1.828	1.850	1.853	1.875
											
	0.286	1.058	1.163	1.772	1.815	1.835	1.865	1.912	1.921	1.949	1.957

CHAPTER 6

CONTEXT SENSITIVE MATCHING

In the previous chapter, we concluded that assigning a priori significance to the shape forming primitives is the primary source of wrong correspondences between two shapes. In this chapter, we explore how we can modify the importance of primitives by putting them in a context. Thus, matching A to B is interpreted as matching the stimulus input A to memory benchmark B, whose category is known. It is assumed that, the general characteristic of category is known and is called *context information*.

A category of objects is a group of similar type of objects. In this study, context information is used to put some restrictions on the possibility of occurrence of shape primitive values. For example, for the *human* category, if the normalized length of the branch that represents the head of the human shape is defined in interval $[l_{min}..l_{max}]$, the normalized length of a possible similar branch must be in this interval. Otherwise, the attribute change cost is increased drastically to penalize this abnormal behavior.

Context information is not only used for restricting shape primitive values but also used to minimize the effect of non-salient branches in a class. Notice that, additional branches in a shape increase the dissimilarity between a shape in the same category, because of the extra insertion and deletion costs.

In this study, since the shape primitives are stored as ordered trees, a common tree is constructed from the shape trees in the same category to find the

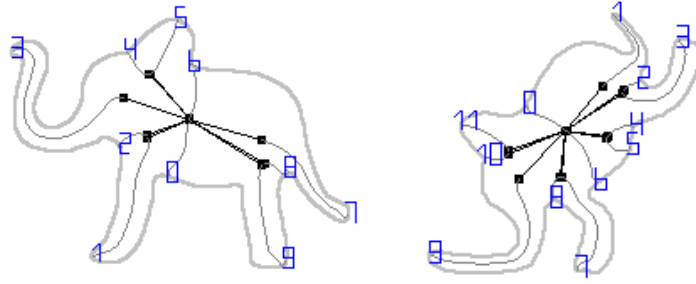


Figure 6.1: Shapes in the same category may contain extra branches

general characteristic of the category. We call this common tree a *shape-category tree*. The main idea behind using shape-category trees in this study is to find possible occurrence intervals for shape primitives. These intervals are taken into account during the comparison of shape primitives in cost calculations. While comparing shape A and shape B with context sensitive matching, the key point is using the shape-category tree of B in cost calculations. Notice that, the category of A is not known.

In this chapter, details of the context sensitive matching and its results are discussed. Note that, the construction process of shape-category trees is a pre-processing for context sensitive matching and because of the multiple description property of the representation, more than one shape-category tree can be constructed for a shape category.

6.1 Construction of Shape-Category Tree

Since a shape category is composed of similar shapes, shape trees of the shapes in the category must also be similar. For example, a human shape is composed of a head, two legs, two arms and a body. All shapes in the *human* category must include at least one of these parts. Due to within category variability or local deformation and articulations, some shapes in the category may contain extra branches as shown in Figure 6.1.

A shape-category tree must hold all the observed and required nodes of each shape tree in the category. Therefore, the category tree construction process must unite the related nodes. Since all shape trees in a category are depth-1, the union of shape trees can be used as the shape-category tree.

To construct a shape-category tree, the shape tree in the category with most populated nodes is chosen as the base-tree. The key idea in the construction process is to find the correspondence between the branches of base-tree and the branches of other trees in the same category. Once a correspondence is found, shape primitives of the non-base node is stored in the corresponding node of the shape-category tree.

Before discussing the pseudo code of the shape-category tree construction process, let us work on an example to clarify the concept. An example category that is composed of three shapes is shown in Figure 6.2. Assume that, all shapes in this category have only one description. For each shape, the attributes are listed at the bottom of the related tree figure.

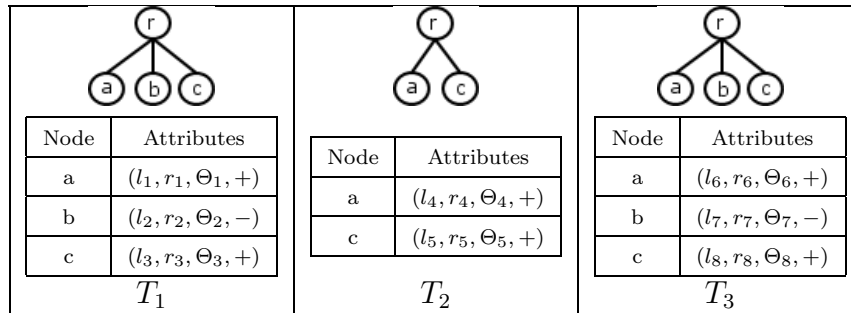


Figure 6.2: A sample shape category

The first step of shape-category tree construction process is the determination of a base-tree. Number of nodes in T_1 and T_3 are equal and greater than the number of nodes in T_2 . Therefore, one of T_1 or T_3 can be chosen as the base-tree. We choose T_1 [the tree with smaller index] as the base-tree.

The second step is the initialization of the shape-category tree. Shape-category tree is a clone of base-tree which stores not attributes but list of attributes in its nodes. In Figure 6.3, the initial category tree of our example is presented. As shown in the figure, the attribute lists are initialized with the attributes of the base-tree.

The third step is the formation of mapping between T_1 and T_2 . In the resultant mapping, node of T_1 labeled as a is mapped to node of T_2 labeled as a and node of T_1 labeled as c is mapped to node of T_2 labeled as c . Since the mapping between T_1 and category tree is one-to-one, the mapping between T_2

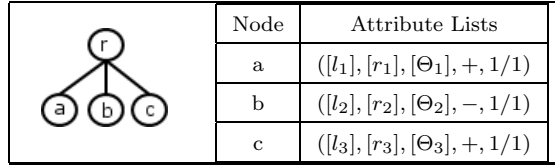


Figure 6.3: Initial shape-category tree of the category in Figure 6.2

and category tree is equivalent to the mapping between T_1 and T_1 . The attributes of nodes of T_2 are inserted into attribute lists of category tree according to the mapping between the nodes of T_2 and category tree. The resultant category tree is shown in Figure 6.4.

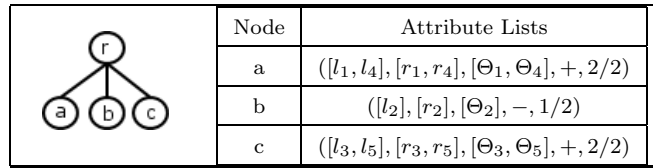


Figure 6.4: Shape-category tree of the category in Figure 6.2 after adding T_2

The last step is finding the mapping between T_1 and T_3 , then, inserting the attributes of nodes of T_3 to attribute lists of category tree. The resultant category tree is shown in Figure 6.5. The important point is that, the final category tree (Figure 6.5) contains all nodes that are contained in the category. Notice that, at the end of attribute lists, a term indicating the frequency of occurrence of branch is added.

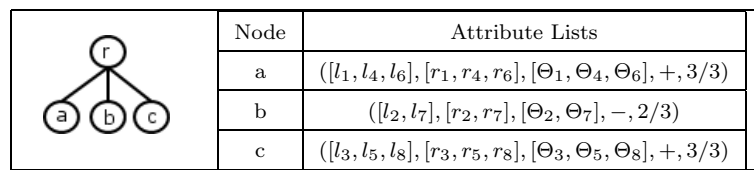


Figure 6.5: Shape-category tree of the category in Figure 6.2 after adding T_3

At this point, the shape-category tree construction process is over, however, multiple descriptions of shapes must be handled. Although only one description exists for all shapes in this example, real life shapes may contain more than one descriptor for any of the shapes in a category. Therefore, the contributor tree descriptions also must be stored in the shape-category tree. In this example, T_1, T_2 and T_3 are stored as the contributor trees.

The following pseudo-code summarizes the shape-category tree construction

process. Let $S_1..S_n$ be a shape category containing n shapes, T_i^j be the i^{th} description of S_j , S_B be the shape whose descriptions are used as base-tree, $\beta(S_j)$ be the number of descriptions of S_j , $|T_i^j|$ be the number of nodes in T_i^j and $CT_1..CT_{\beta(S_B)}$ be the list of resultant category trees.

Algorithm *ConstructCategoryTreeList()*

Input: Shape class ($S_1..S_n$)

Output: List of category trees ($CT_1..CT_{\beta(S_B)}$)

(* Find the first most populated shape *)

1. $B \leftarrow 1$

2. **for** $j \leftarrow 1$ **to** n

3. **if** $|T_i^j| > |T_i^B|$

4. $B \leftarrow j$

5.

(* Create category trees *)

6. **for** $i \leftarrow 1$ **to** $\beta(S_B)$

7. insert attributes of T_i^B to CT_i

8.

(* Find mappings with other trees and complete construction *)

9. **for** $k \leftarrow 1$ **to** $\beta(S_B)$

10. **for** $j \leftarrow 1$ **to** n

11. **if** $j \neq B$

12. $cost \leftarrow \infty$

13. $index \leftarrow -1$

14. **for** $i \leftarrow 1$ **to** $\beta(S_B)$

15. $temp \leftarrow editDistance$ between T_k^B and T_i^j

16. **if** $temp < cost$

17. $cost \leftarrow temp$

18. $index \leftarrow i$

19. Insert attributes of T_{index}^j to T_k^B

20. Insert T_{index}^j as contributor to T_k^B

The category tree generated for the **hand** category is shown in Figure 6.6, for illustrative purposes. Gaussians are fit to disconnection points and the boundary of one of the shapes in this category is drawn.

To conclude, just like shape trees, category trees are ordered and depth-1. The difference lies in the attributes. Each edge (inclusion property) of a category

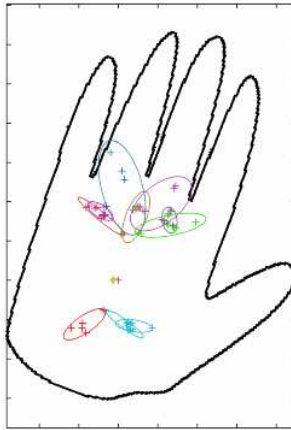


Figure 6.6: Category tree for the hand.

tree carry an attribute denoting the probability that the edge exists in a random shape in that particular category. Unlike scalar-valued nodal attributes of a shape tree, nodal attributes of a category tree are vector-valued, reflecting the observed distribution.

6.2 Using Shape-Category Trees In Matching

Let B be a shape in category k and A be a shape that is queried to find a similarity measure with B . If shape A is similar to shape B , it must be also similar to the other members of the category k . Therefore, attribute lists that are stored in shape-category tree of k guides the matching between shape A and shape B . The process is shown in Figure 6.7.

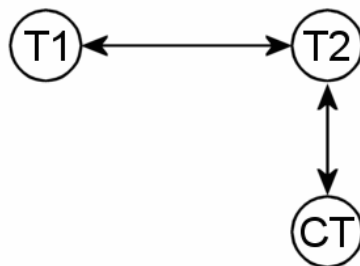


Figure 6.7: Using shape-category tree in matching

As discussed before, shape-category tree construction process may lead to more than one category tree for a category. Each category tree construction process has a construction cost which is the summation of editing distance costs

between base-tree and contributor trees of the category tree. Because of indentations and articulations, the shape whose descriptions are used as base-tree may have more descriptions than other shapes in the same class. This behavior leads to unequal construction costs for shape-category trees. The category tree of a class with minimum construction cost is called *best shape-category tree*. In matching process of a queried shape and a benchmark shape, for efficiency, all descriptions of queried shape are matched with only the description of the benchmark shape which is a contributor tree to the best shape-category tree.

6.2.1 Context Guided Editing Operation Costs

The matching of two shapes is the process of matching two shape trees. Let the i^{th} node of T_k be associated with the j^{th} node of the category tree $T_{c(k)}^{cat}$ where $c(k)$ denotes the category to which shape k may belong. Then a node in $T_{c(k)}^{cat}$ holds information related to n nodes from m shape trees forming the category tree ($n \leq m$). The observed range values for r , θ , l of the j^{th} node of $T_{c(k)}^{cat}$ are $[r_{min} \dots r_{max}]_j$, $[\theta_{min} \dots \theta_{max}]_j$ and $[l_{min} \dots l_{max}]_j$.

Let T_1 be one of the shape trees of the queried shape and T_2 be the shape tree of the benchmark object. Matching of T_1 and T_2 can be guided or influenced by the category tree $T_{c(k)}^{cat}$. For that purpose, we have defined a generic function $f(x, y, [min, max])$ which calculates a cost in a given range (see Figure 6.8). In the experiments, we take $\theta_1 = \frac{\pi}{4}$ and $\theta_2 = \frac{2\pi}{9}$.

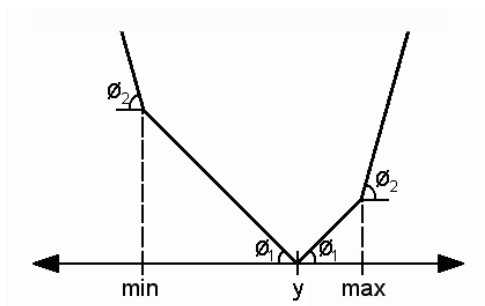


Figure 6.8: The cost function $f(x, y, [min, max])$ used in prior-guided matching.

CHANGE_COST which is the modified cost of changing the attributes of the i^{th} node of T_1 to the attributes of the j^{th} node of T_2 with the assumption that both nodes are associated to the k^{th} node of the category tree $T_{c(2)}^{cat}$ is given by:

$$\frac{3 \times f(l_i^1, l_j^2, [l_{min}, l_{max}]_k) + f(r_i^1, r_j^2, [r_{min}, r_{max}]_k) + f(\theta_i^1, [\theta_{min}, \theta_j^2, \theta_{max}]_k)}{5} \times \frac{n}{m}$$

where $\frac{n}{m}$ can be interpreted as the frequency term which reflects the prominent strongness of the primitive.

Similarly, deletion and insertion costs are defined as follows:

$$\text{DELETION_COST} = (1 - r_i^1) \times \frac{l_i^1 \times \text{DEL_COST}}{l_{max}^1}$$

$$\text{INSERTION_COST} = (1 - r_j^2) \times \frac{l_j^2 \times \text{INS_COST}}{l_{max}^2} \times \frac{n}{m}$$

Note that we always apply operations to T_1 to morph it to T_2 . Hence, deletion implies deleting a primitive from T_1 whereas insertion implies inserting a node of T_2 to T_1 which does not exist in T_1 . This fact is reflected in the cost definitions.

6.3 Experiments

To test the approach, the retrieval problem is used. Each shape in the image database is used as query shape and compared with all other shapes, except itself. Note that, the query shape is not used in any shape category tree construction. The results are presented in Table 6.2. According to Bull's eye test [25, 10], the retrieval rate is 99.111% for top 12 matches. The retrieval rate decreases to 97.44%, if top 6 matches are taken into account.

When the query shape is very different from other shapes in the same category, the retrieval lives trouble, as shown in Figure 6.9. In Table 6.1, the normalized lengths of tails of crocodiles are presented. Observe that, the tail of second crocodile (query shape in Figure 6.9) is smaller than others. Because of the high penalizing to primitives that do not lie in the possible occurrence interval determined by category, outlier primitives cause problems. This problem occurs for the categories that have shapes containing primitives with low variability. Notice the continues dissimilarity measures in Figure 6.9. For successful retrievals, the dissimilarity measures are not continues (see Table 6.2), i.e. the

	2.588	2.604	2.612	2.671	2.676	2.678	2.705	2.751	2.751	2.794	2.813

Figure 6.9: An outlier shape in the category

0.5487	0.4730	0.6368	0.5492	0.5502	0.5311

Table 6.1: Normalized lengths of tails of crocodiles

dissimilarity measures belonging to shapes from other categories are relatively high.

When a shape containing outlier primitives is refused by its own category, it may be accepted by another category that have shapes containing primitives with high variability. A sample situation is illustrated in Figure 6.10.

	1.922	2.094	2.133	2.207	2.227	2.292	2.363	2.427	2.431	2.484	2.490

Figure 6.10: A shape accepted by another category

A shape in a category may be more similar to some shapes in other categories, as illustrated in Figure 6.11. Observe that, for such shapes, the dissimilarity measures do not contain strong jumps for similar categories.

	1.434	1.505	1.539	1.539	1.616	1.618	1.646	1.659	1.719	1.832	1.880
	2.056	2.085	2.196	2.263	2.309	2.322	2.627	2.726	2.751	2.768	3.353

Figure 6.11: Structurally similar shapes from different categories

As seen in Table 6.2, the dissimilarity measures are not symmetric. That is, if the dissimilarity measure between shape A and shape B is α , the dissimilarity measure between shape B and shape A does not need to be α . The reason of this behavior is that the used shape-category trees for matching between $A-B$ and $B-A$ may be different. During the matching process of $A-B$, the shape-category tree of B restricts the attribute intervals and determines the frequency terms. But, during the matching process of $B-A$, the shape-category tree of A guides

the matching. Therefore, the context sensitive matching is not symmetric.

Table 6.2: Retrieval results






























































































































































































































												
	0.298	0.345	0.348	0.354	0.378	2.440	2.472	2.482	2.507	2.537	2.630	
												
	0.478	0.517	0.607	0.609	0.622	2.458	2.519	2.529	2.537	2.550	2.554	
												
	0.337	0.396	0.404	0.504	0.560	2.574	2.601	2.685	2.686	2.698	2.788	
												
	0.228	0.358	0.437	0.455	0.463	2.399	2.418	2.440	2.484	2.560	2.566	
												
	0.534	0.633	0.684	0.708	0.763	2.643	2.657	2.669	2.722	2.749	2.960	
												
	0.355	0.491	0.578	0.637	0.645	2.394	2.407	2.408	2.495	2.518	2.596	
												
	1.420	1.512	1.597	1.610	1.635	2.957	2.976	3.019	3.052	3.086	3.093	
												
	0.650	0.687	0.693	0.696	0.737	2.777	2.815	2.846	2.864	2.890	2.937	
												
	1.434	1.505	1.539	1.539	1.616	1.618	1.646	1.659	1.719	1.832	1.880	
												
	0.715	0.882	0.928	1.030	1.097	2.361	2.444	2.559	2.585	2.597	2.622	
												
	0.641	0.735	0.852	1.036	1.042	1.967	2.078	2.162	2.215	2.268	2.312	
												
	0.952	1.071	1.112	1.155	1.300	1.340	1.393	1.401	1.496	1.519	1.555	
												
	0.375	0.438	0.451	0.596	0.722	2.122	2.132	2.149	2.184	2.190	2.197	
												
	0.551	0.621	0.625	0.876	1.000	2.100	2.129	2.129	2.170	2.203	2.213	
												
	0.670	0.744	0.748	1.026	1.270	2.456	2.505	2.505	2.553	2.573	2.595	
												
	0.725	0.806	0.878	0.908	0.908	2.207	2.252	2.357	2.366	2.392	2.426	
												
	1.941	2.082	2.188	2.193	2.250	2.265	2.300	2.309	2.327	2.344	2.344	

Table 6.2 (continued)













































































































































































































											
	0.494	0.560	0.572	0.858	1.011	2.394	2.422	2.494	2.515	2.540	2.540
											
	0.727	0.815	0.845	0.990	1.081	2.097	2.137	2.212	2.258	2.267	2.430
											
	0.410	0.450	0.466	0.715	0.796	2.577	2.622	2.682	2.686	2.778	2.903
											
	0.426	0.848	0.849	0.974	1.113	1.943	2.051	2.060	2.061	2.123	2.337
											
	1.103	1.129	1.205	1.615	1.722	2.355	2.477	2.535	2.542	2.626	2.678
											
	0.632	0.714	0.760	1.062	1.124	2.078	2.121	2.124	2.198	2.238	2.378
											
	0.835	1.339	1.348	1.446	1.628	2.238	2.325	2.327	2.373	2.379	2.757
											
	0.609	0.906	0.949	1.028	1.435	2.935	2.962	2.987	3.004	3.005	3.009
											
	0.457	0.806	0.835	1.056	1.265	2.958	2.962	2.967	2.992	3.029	3.048
											
	1.067	1.107	1.170	1.288	1.442	2.553	2.585	2.659	2.665	2.706	2.754
											
	1.034	1.059	1.068	1.244	1.298	3.471	3.553	3.600	3.614	3.625	3.628
											
	1.176	1.269	1.289	1.311	1.706	3.338	3.348	3.356	3.404	3.415	3.415
											
	2.119	2.301	2.366	2.372	2.481	2.869	2.955	2.986	3.105	3.177	3.195
											
	0.437	0.657	0.709	0.829	0.940	1.872	1.963	1.994	2.012	2.169	2.231
											
	0.672	0.969	1.070	1.155	1.187	1.711	1.801	1.868	1.975	1.983	2.202
											
	2.046	2.085	2.197	2.417	2.578	2.779	2.810	2.904	2.910	2.951	3.311
											
	0.258	0.290	0.532	0.570	0.665	2.005	2.131	2.154	2.226	2.254	2.292

Table 6.2 (continued)

	0.322	0.339	0.471	0.618	0.714	1.667	1.713	1.793	1.828	1.858	1.932	
	1.287	1.421	1.558	1.577	1.971	1.980	2.029	2.052	2.165	2.247	2.359	
	0.382	0.441	0.589	0.606	1.087	1.284	1.297	1.300	1.300	1.368	1.389	
	0.288	0.357	0.425	0.542	0.773	1.768	1.809	1.814	1.833	1.894	1.977	
	0.286	0.295	0.547	0.565	0.643	1.641	1.647	1.705	1.718	1.732	1.740	
	2.448	2.549	2.576	2.710	2.736	3.026	3.074	3.088	3.103	3.149	3.249	
	0.926	0.973	1.022	1.050	1.434	2.085	2.098	2.153	2.156	2.167	2.328	
	1.099	1.132	1.170	1.195	1.535	2.167	2.189	2.212	2.214	2.312	2.349	
	0.208	0.248	0.474	0.474	0.489	2.690	2.716	2.815	2.826	2.845	2.872	
	0.124	0.150	0.365	0.376	0.403	2.464	2.487	2.512	2.590	2.591	2.618	
	0.054	0.056	0.284	0.309	0.336	2.499	2.546	2.571	2.626	2.626	2.632	
	0.123	0.152	0.362	0.362	0.382	2.777	2.784	2.860	2.908	2.924	2.926	
	0.177	0.188	0.362	0.375	0.376	2.731	2.787	2.823	2.838	2.840	2.898	
	0.151	0.180	0.393	0.426	0.448	2.545	2.547	2.574	2.629	2.634	2.654	
	0.912	1.105	1.147	1.326	1.337	2.561	2.601	2.605	2.734	3.079	3.104	
	1.503	1.503	1.520	1.594	1.649	2.549	2.627	2.658	2.690	2.701	2.762	
	0.442	0.542	0.611	0.703	0.746	2.707	2.724	2.836	2.865	2.871	2.943	

Table 6.2 (continued)

	0.484	0.567	0.607	0.778	0.843	2.815	2.816	2.830	3.030	3.037	3.195
	1.346	1.392	1.405	1.421	1.506	1.764	1.827	1.847	1.892	2.226	2.690
	0.555	0.615	0.766	0.799	0.841	2.472	2.473	2.478	2.733	2.751	2.764
	1.173	1.432	1.581	1.599	1.628	1.636	1.717	1.751	1.769	1.788	1.852
	0.288	0.772	0.848	0.927	0.944	1.346	1.359	1.396	1.401	1.508	1.602
	0.589	0.650	0.898	0.903	1.033	1.816	1.826	1.888	1.895	1.949	2.058
	1.161	1.429	1.544	1.557	1.598	1.918	1.978	2.045	2.060	2.143	2.190
	0.526	0.572	0.768	0.770	0.823	0.997	1.146	1.164	1.171	1.239	1.274
	0.217	0.799	0.802	0.898	1.104	3.022	3.059	3.114	3.193	3.256	3.358
	0.613	0.711	0.712	0.892	0.981	3.569	3.600	3.710	3.812	3.904	4.001
	0.637	0.726	0.729	0.976	0.999	2.869	2.946	3.074	3.143	3.158	3.261
	0.172	0.438	0.447	0.633	0.763	3.412	3.470	3.564	3.572	3.593	3.616
	1.432	1.437	1.481	1.485	1.492	3.761	3.776	3.814	3.881	3.886	3.892
	0.318	0.587	0.594	0.827	0.933	3.155	3.282	3.352	3.397	3.424	3.473
	1.782	1.829	1.843	1.914	1.917	2.857	2.860	2.872	2.993	3.011	3.035
	0.399	0.442	0.494	0.573	0.578	2.556	2.589	2.828	2.852	2.860	2.946
	0.651	0.654	0.780	0.843	0.880	3.010	3.043	3.085	3.111	3.154	3.163

Table 6.2 (continued)

	0.613	0.652	0.814	0.875	0.887	2.855	2.873	2.922	2.930	2.932	2.965
	0.388	0.420	0.493	0.566	0.600	2.769	2.779	2.793	2.814	2.833	2.877
	0.073	0.207	0.392	0.449	0.480	2.724	2.731	2.791	2.825	2.902	3.009
	0.181	0.348	0.505	0.595	0.601	2.600	2.603	2.677	2.712	3.013	3.060
	0.109	0.128	0.295	0.318	0.385	2.190	2.228	2.229	2.245	2.350	2.444
	0.046	0.091	0.266	0.302	0.384	2.464	2.485	2.560	2.576	2.625	2.723
	1.059	1.067	1.079	1.256	1.381	2.316	2.319	2.328	2.331	2.444	2.457
	0.552	0.554	0.580	0.706	0.831	2.474	2.490	2.663	2.780	2.784	2.832
	1.491	1.507	1.515	1.605	1.702	2.332	2.390	2.613	2.815	2.855	2.900
	0.101	0.166	0.313	0.383	0.447	2.486	2.507	2.593	2.609	2.646	2.792
	0.470	0.567	0.747	0.783	0.812	2.548	2.567	2.568	2.611	2.730	2.778
	0.171	0.437	0.672	0.709	0.709	2.781	2.806	2.824	2.836	2.891	2.904
	0.212	0.279	0.520	0.547	0.617	2.851	2.863	2.866	2.928	2.987	3.029
	0.207	0.478	0.711	0.715	0.761	2.734	2.763	2.790	2.795	2.837	2.844
	0.405	0.570	0.736	0.750	0.875	2.777	2.780	2.797	2.883	2.887	2.947
	1.161	1.166	1.394	1.398	1.458	3.159	3.171	3.196	3.226	3.308	3.317
	0.378	0.440	0.458	0.673	0.817	2.894	2.901	2.909	2.951	2.978	3.005

Table 6.2 (continued)

	0.950	1.076	1.132	1.224	1.299	2.701	2.810	2.866	2.881	2.971	3.069	
	0.461	0.511	0.677	0.736	0.941	2.461	2.521	2.558	2.581	2.599	2.618	
	1.870	1.896	1.912	1.935	1.962	1.970	1.980	1.985	2.066	2.092	2.160	
	1.162	1.294	1.328	1.394	1.615	2.753	2.759	2.764	2.795	2.803	2.888	
	0.359	0.422	0.575	0.784	0.785	2.654	2.700	2.715	2.748	2.792	2.831	
	1.327	1.337	1.380	1.474	1.536	2.420	2.451	2.484	2.549	2.551	2.580	
	0.804	0.989	1.040	1.055	1.177	2.556	2.565	2.571	2.585	2.616	2.625	
	0.227	0.441	0.587	0.607	0.658	2.395	2.420	2.446	2.459	2.522	2.609	
	0.680	0.681	0.815	0.880	0.884	2.206	2.211	2.255	2.274	2.282	2.333	
	0.417	0.531	0.573	0.601	0.640	2.318	2.435	2.467	2.477	2.490	2.499	
	0.156	0.372	0.466	0.505	0.603	2.386	2.412	2.431	2.450	2.507	2.604	
	1.215	1.227	1.256	1.281	1.292	2.279	2.279	2.348	2.376	2.400	2.435	
	0.482	0.495	0.525	0.533	0.553	2.303	2.307	2.336	2.359	2.365	2.407	
	0.301	0.364	0.379	0.399	0.406	2.122	2.181	2.182	2.195	2.197	2.212	
	0.347	0.403	0.466	0.550	0.561	2.245	2.290	2.302	2.397	2.425	2.436	
	0.180	0.265	0.273	0.317	0.340	2.247	2.326	2.365	2.411	2.416	2.428	
	0.519	0.556	0.564	0.640	0.659	2.685	2.693	2.728	2.753	2.754	2.778	

Table 6.2 (continued)

	0.247	0.596	0.597	0.699	1.156	2.877	2.908	2.945	2.962	2.962	3.011
	2.588	2.604	2.612	2.671	2.676	2.678	2.705	2.751	2.751	2.794	2.813
	1.127	1.285	1.291	1.411	1.545	2.350	2.530	2.593	2.663	2.677	2.717
	0.962	0.972	1.088	1.185	1.395	2.578	2.669	2.738	2.804	2.846	2.916
	0.197	0.537	0.554	0.655	1.130	2.931	2.937	2.963	2.967	2.991	2.991
	1.678	1.722	1.839	1.983	2.414	3.248	3.289	3.352	3.366	3.382	3.393
	0.749	0.812	0.886	0.979	0.979	2.069	2.170	2.185	2.213	2.321	2.361
	0.267	0.395	0.570	0.646	0.646	1.801	1.829	1.859	2.080	2.121	2.138
	0.213	0.213	0.460	0.473	0.498	1.921	1.951	1.954	1.977	1.995	2.021
	0.344	0.535	0.622	0.711	0.711	1.855	1.892	1.937	2.038	2.093	2.129
	0.000	0.237	0.577	0.586	0.599	1.947	1.997	1.999	2.001	2.112	2.112
	0.000	0.237	0.577	0.586	0.599	1.947	1.997	1.999	2.001	2.112	2.112
	0.561	0.587	0.611	0.778	1.050	3.258	3.374	3.383	3.447	3.479	3.592
	0.753	0.794	0.976	0.996	1.208	3.306	3.418	3.463	3.565	3.625	3.680
	0.793	0.910	0.968	0.989	1.177	3.118	3.148	3.223	3.305	3.324	3.351
	1.999	2.209	2.252	2.287	2.380	3.026	3.064	3.274	3.428	3.494	3.553
	0.980	0.981	0.996	1.022	1.038	3.265	3.302	3.346	3.370	3.390	3.399

Table 6.2 (continued)

	0.902	0.912	1.006	1.121	1.444	3.402	3.527	3.545	3.565	3.569	3.598
	0.231	0.255	0.272	0.546	1.162	2.258	2.362	2.459	2.724	2.728	2.769
	1.878	1.885	1.886	1.936	2.239	2.375	2.380	2.414	2.538	2.599	2.624
	0.165	0.225	0.234	0.584	0.856	2.314	2.426	2.517	2.865	2.937	2.983
	0.208	0.273	0.293	0.719	0.941	2.334	2.462	2.617	2.741	2.765	2.772
	1.508	1.562	1.567	1.574	1.877	2.678	2.775	2.817	2.855	2.872	2.985
	0.186	0.265	0.275	0.678	0.848	2.195	2.294	2.459	2.798	2.860	2.917
	0.234	0.277	0.293	0.294	0.748	2.725	2.776	2.786	2.817	2.884	2.904
	0.078	0.137	0.180	0.332	0.761	2.685	2.759	2.764	2.783	2.849	2.864
	0.425	0.518	0.547	0.606	0.759	2.913	2.948	2.996	2.999	3.066	3.087
	0.117	0.152	0.262	0.400	0.833	2.685	2.756	2.768	2.781	2.847	2.864
	2.667	2.694	2.709	2.728	2.756	3.379	3.393	3.401	3.416	3.565	3.578
	0.298	0.322	0.425	0.605	1.014	2.708	2.786	2.790	2.805	2.871	2.888
	1.715	2.137	2.145	2.328	2.395	4.127	4.193	4.261	4.265	4.380	4.401
	1.008	1.038	1.234	1.615	1.843	2.107	2.243	2.327	2.432	2.523	2.813
	2.056	2.085	2.196	2.263	2.309	2.322	2.627	2.726	2.751	2.768	3.353
	0.727	0.757	1.025	1.034	1.357	3.573	3.669	3.734	3.837	3.844	3.920

Table 6.2 (continued)













































































































































































































											
	0.629	0.872	0.957	1.185	1.367	1.856	1.870	1.916	2.075	2.102	2.339
											
	2.158	2.179	2.208	2.649	2.855	3.487	3.489	3.592	3.608	3.794	3.875
											
	1.237	1.606	1.722	1.799	1.917	2.923	2.960	2.979	2.985	2.998	3.031
											
	1.501	1.599	1.697	1.932	2.345	2.676	2.779	3.048	3.109	3.110	3.133
											
	0.628	0.666	0.810	1.286	1.540	2.567	2.600	2.633	2.713	2.715	2.767
											
	0.492	0.752	0.994	1.477	1.711	2.489	2.490	2.540	2.585	2.619	2.626
											
	0.548	0.770	1.148	1.566	1.934	2.433	2.523	2.634	2.649	2.659	2.659
											
	1.785	2.606	2.643	2.801	2.844	3.600	3.625	3.707	3.724	3.725	3.767
											
	0.142	0.156	0.260	0.426	0.449	2.097	2.181	2.224	2.278	2.301	2.444
											
	0.246	0.246	0.271	0.297	0.314	2.345	2.438	2.480	2.556	2.566	2.588
											
	0.746	0.767	0.875	0.889	0.891	2.351	2.452	2.470	2.613	2.726	2.759
											
	0.210	0.221	0.338	0.464	0.505	2.284	2.376	2.408	2.538	2.555	2.661
											
	0.125	0.150	0.242	0.392	0.419	2.186	2.278	2.317	2.402	2.421	2.535
											
	0.577	0.581	0.680	0.681	0.697	2.634	2.717	2.724	2.743	2.802	2.880
											
	0.333	0.861	0.948	1.004	1.389	2.375	2.383	2.391	2.490	2.503	2.579
											
	3.732	3.747	3.768	3.803	3.822	3.841	4.032	4.047	4.052	4.056	4.071
											
	0.156	0.715	0.806	0.871	1.267	2.303	2.309	2.324	2.412	2.434	2.482

Table 6.2 (continued)





































































































































































































































































































































											
	1.061	1.085	1.498	1.505	1.816	3.368	3.523	3.533	3.596	3.600	3.604
											
	0.253	0.484	0.831	0.848	1.108	3.457	3.510	3.511	3.646	3.688	3.695
											
	0.226	0.433	0.714	0.734	1.196	3.429	3.437	3.465	3.484	3.618	3.666
											
	0.084	0.273	1.012	1.284	1.307	2.928	2.939	2.955	2.987	3.008	3.082
											
	0.091	0.246	1.086	1.359	1.385	2.724	2.757	2.789	2.818	3.096	3.135
											
	0.632	0.822	1.602	1.613	1.653	2.331	2.331	2.385	2.388	2.483	2.511
											
	0.466	0.607	1.412	1.425	1.461	2.046	2.077	2.132	2.134	2.229	2.249
											
	0.621	0.656	1.288	1.554	1.576	3.025	3.029	3.117	3.128	3.152	3.158
											
	0.412	0.461	0.945	0.964	0.993	2.423	2.472	2.495	2.504	2.510	2.558
											
	0.076	0.119	0.355	0.391	0.592	2.076	2.191	2.212	2.219	2.224	2.271
											
	0.383	0.574	0.587	0.630	0.891	2.390	2.528	2.555	2.610	2.637	2.713
											
	0.120	0.139	0.411	0.450	0.708	2.240	2.321	2.342	2.348	2.371	2.386
											
	1.922	2.094	2.133	2.207	2.227	2.292	2.363	2.427	2.431	2.484	2.490
											
	0.388	0.615	0.629	0.679	0.857	2.518	2.683	2.711	2.723	2.747	2.766
											
	0.325	0.349	0.641	0.685	0.921	2.121	2.147	2.168	2.204	2.218	2.267
											
	0.263	0.691	0.874	0.933	1.211	2.038	2.088	2.092	2.118	2.148	2.297
											
	0.615	1.216	1.249	1.301	1.530	1.632	1.664	1.668	1.682	1.695	1.896

Table 6.2 (continued)

											
	1.412	1.741	1.749	1.764	1.801	2.552	2.646	2.652	2.707	2.724	2.767
											
	0.912	0.919	0.958	1.008	1.189	2.608	2.754	2.772	2.778	2.827	2.831
											
	0.399	0.969	1.065	1.092	1.322	2.874	2.990	3.018	3.019	3.040	3.045
											
	0.164	0.758	0.861	0.999	0.999	2.919	2.924	2.974	2.976	2.982	3.023
											
	0.222	0.712	0.773	0.786	0.793	2.522	2.537	2.689	2.767	2.810	2.867
											
	0.268	0.928	0.954	1.003	1.026	2.419	2.597	2.617	2.683	2.699	2.712
											
	0.261	0.631	0.667	1.062	1.063	2.496	2.507	2.586	2.598	2.598	2.610
											
	0.166	0.834	0.847	0.910	0.924	2.486	2.652	2.671	2.731	2.749	2.761
											
	0.253	0.625	0.653	0.999	1.008	2.596	2.619	2.649	2.659	2.712	2.747
											
	0.199	0.576	0.579	0.949	0.954	2.459	2.471	2.635	2.722	2.746	2.823

CHAPTER 7

CONCLUSION

In this thesis, calculation of dissimilarity between two shapes using skeletal trees has been presented. After converting shapes into depth-1 shape trees, a tree editing distance algorithm is used to find the dissimilarity between the shapes. Shape primitives and their spatial organizations are used to find the cost to morph one shape tree into another. As discussed in detail, assigning a priori significance to shape primitives may cause wrong correspondence between two shapes.

To modify the importance of primitives, they are put in context. In this study, characteristics of shape categories are used as context information. To find the dissimilarity between shape A and shape B, category characteristics of shape B guide the cost calculations. Recall that, characteristics of a category are obtained from shape-category trees that are constructed by uniting the shape trees in a category.

The shape retrieval problem is used for experiments. When dissimilarity calculations are done with the guidance of context, better retrieval results are obtained. Also, the obtained dissimilarity measures violate the symmetry metric. That is, the dissimilarity between A and B is not equal to dissimilarity between B and A. This behavior is supported by experimental results of [14, 15] and a natural result of using the guidance of different category trees to compare A-B and B-A.

Hence the shape category trees are also depth-1, it is possible to compare a shape with a category and a category with a category. Therefore, as a future work, it is possible to use the shape category trees for classification. Also, the proposed context sensitive approach can be used for clustering, if a technique that uses asymmetric similarity matrix is developed.

REFERENCES

- [1] D. Jacobs, D. Weinshall, and Y. Gdalyahu. Classification with nonmetric distances: Image retrieval and class representation, 2000.
- [2] C. Aslan and S. Tari. An axis-based representation for recognition. In *ICCV*, pages 1339–1346, 2005.
- [3] C. Aslan. Disconnected skeletons for shape recognition. Master’s thesis, Department of Computer Engineering, Middle East Technical University, May 2005.
- [4] K. Siddiqi, A. Shokoufandeh, S. J. Dickinson, and S. W. Zucker. Shock graphs and shape matching. *Int. J. Comput. Vision*, 35(1):13–32, 1999.
- [5] T.L. Liu and D. Geiger. Approximate tree matching and shape similarity. In *ICCV*, pages 983–1001, 1998.
- [6] S. C. Zhu and A. L. Yuille. Forms: a flexible object recognition and modeling system. *Int. J. Comput. Vision*, 20(3):187–212, 1996.
- [7] T. B. Sebastian, P. N. Klein, and B. B. Kimia. Recognition of shapes by editing their shock graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(5):550–571, 2004.
- [8] B. B. Kimia, A. R. Tannenbaum, and S. W. Zucker. Shapes, shocks, and deformations i: the components of two-dimensional shape and the reaction-diffusion space. *Int. J. Comput. Vision*, 15(3):189–224, 1995.
- [9] S. Tari, J. Shah, and H. Pien. Extraction of shape skeletons from grayscale images. *CVIU*, 66(2):133–146, 1997.
- [10] L. J. Latecki and R. Lakamper. Shape similarity measure based on correspondence of visual parts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(10):1185–1190, 2000.
- [11] A. Imiya and U. Eckhardt. Discrete mean curvature flow. In *Scale-Space*, pages 477–482, 1999.
- [12] D. Shasha Zhang, K. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Computing*, 18:1245–1262, 1989.

- [13] R. Basri, L. Costa, D. Geiger, and D. Jacobs. Determining the similarity of deformable shapes. Technical Report CS98-05, 1, 1998.
- [14] A. Tversky. Features of similarity. *Psychological Review*, 84:327–352, 1977.
- [15] S. Kosslyn D. Mumford, R. Herrnstein and W. Vaughan. Analysis and synthesis of human and avian categorizations of 15 simple polygons. *preprint, Harvard University Dept. of Psych*, 1989.
- [16] D. Mumford. Mathematical theories of shape: Do they model perception? In *Proc. Geometric Methods in Computer Vision Conference, SPIE*, volume 1570, pages 2–10, 1991.
- [17] S. Edelman, F. Cutzu, and S. Duvdevani-Bar. Similarity to reference shapes as a basis for shape representation, 1996.
- [18] A. Torsello and E. R. Hancock. Matching and embedding through edit-union of trees. In *ECCV (3)*, pages 822–836, 2002.
- [19] L. Ambrosio and V. Tortorelli. On the approximation of functionals depending on jumps by elliptic functionals via Γ -convergence. *Commun. Pure Appl. Math.*, 43(8):999–1036, 1990.
- [20] D. Mumford and J. Shah. Optimal approximations by piecewise smooth functions and associated variational problems. *Commun. Pure Appl. Math.*, 42(5):577–685, 1989.
- [21] M.J. Fischer R.A. Wagner. The string-to-string correction problem. *Journal of the ACM*, 21:168–173, 1974.
- [22] S.M Selkow. The tree-to-tree editing problem. *Information Processing Letters*, 6:184–186, 1977.
- [23] K.C Tai. The tree-to-tree correction problem. *J. ACM*, 26:422–433, 1979.
- [24] P. Klein. Computing the edit-distance between unrooted ordered trees. In *Proceedings of the 6th Annual European Symposium*, number 1461, pages 91–102, 1998.
- [25] H. Ling and D. W. Jacobs. Using the inner-distance for classification of articulated shapes. In *CVPR*, pages 719–726, 2005.
- [26] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.