Indexing and Matching for View-Based 3-D Object Recognition Using Shock Graphs

by

Diego Alejandro Macrini

A thesis submitted in conformity with the requirements
for the degree of Master's in Computer Science
Graduate Department of Computer Science
University of Toronto

Abstract

Indexing and Matching for View-Based 3-D Object Recognition Using Shock Graphs

Diego Alejandro Macrini

Master's in Computer Science

Graduate Department of Computer Science

University of Toronto

2003

A shock graph is a shape abstraction that decomposes a shape into hierarchically organized primitive parts [55]. In this thesis, we propose a novel shock graph computation algorithm that yields stable graphs under noise and shape deformation due to viewpoint changes. Next, we extend the indexing and matching framework for hierarchical structures introduced by Shokoufandeh et al. [54], and apply it to the problem of matching shock graphs representing views of 3-D objects. The indexing performance is improved by a vote accumulation algorithm that efficiently solves a multiple one-to-one assignment of votes. In turn, the matching algorithm is extended by ensuring the satisfaction of all the hierarchical constraints encoded in the graphs. We show that the proposed integrated framework is effective in recognizing object shapes and in estimating the pose of their corresponding 3-D object models. We demonstrate the performance of the framework with a database of 2688 object views.

To my family.

Acknowledgements

I would like to begin by expressing my thanks to my supervisor, Sven Dickinson, for his great support and dedication. I would also like to thank my second reader, Allan Jepson, for his time and detailed revision of the thesis.

I am grateful to all the vision group for creating a stimulating environment for discussion. In particular, I thank Chakra Chennubhotla for his openness to sharing his ideas. Special thanks also go to John Midgley, the lab is no longer as much fun without him; Faisal Zubair Qureshi, whose positive humor is contagious; Francisco Estrada; and Gustavo Carneiro for always being interested in expressing their views.

I am specially grateful to Ali Shokoufandeh for his help and prompt feedback. Furthermore, I would like to thank a number of people that participated in some way or another in the development of the DAG matcher. Without doubt, the most important contribution is that of Maxim Trokhimtchouk who developed a useful application to capture views of 3-D CAD models and assembled the nearest-neighbour search database for our experiments. Other students contributed with insightful comments and with testing the system for different tasks; among them, I would like to mention Chinh Khac Nguyen, Neil Witcomb, Michel-Alexandre Cardin, and Jordan Hesse.

The journey of working on this thesis has allowed me to meet two excellent vision groups in other universities. In both cases, I was treated better than how I could have possibly expected. Firstly, I thank Kaleem Siddiqi and his students at McGill University for their time in answering my questions. In particular, I thank Pavel Dimitrov for allowing us to use his code to compute skeletons and Sylvain Bouix for being a wonderful host. Secondly, my sincere thanks go to Bart M. ter Haar Romeny and his group at the Eindhoven University of Technology for inviting Sven and I, and for adopting us as part of their group during our visit. My special thanks go to Markus van Almsick and his lovely family for being such caring hosts during my brief visit to Germany.

I would like to thank all of my friends who have been supporting me on a daily basis:

Andre Paes Dasilva, Patricio Simari, Josh Buresh-Oppenheim, Sebastian Sardiña, Blas Melissari, Pablo Sala, Ariel Fuxman, Flavio Rizzolo, and Alfredo Gabaldon. You have made me feel at home!

The support and encouragement of my family are a wonderful treasure for which I am most grateful. Thank you Mamá, Papá, Desirée, Luqui, Alejandra, abuelas y tías!

There is one person who supported me in a way that I would never had imagined and whom I feel enormously fortunate to have met. I thank you Luisa for your support, understanding, editing, and for sharing with me your own experience in writing a thesis.

Last, and least as well, I must mention the boxer, a faithful dog that you will soon meet if continuing reading. I have come to know this dog all the way to its bones, and thus I thank its anonymous creator for this contribution.

Contents

Chapter 1

Introduction

One of the most interesting features of the human visual system is its ability to quickly recognize a stimulus from a large visual memory. What is truly remarkable is that this efficient mechanism works even with objects that are only partly similar to objects in memory. This function of the brain is not fully understood, but something seems to be clear: the brain does not compare the stimuli with all the objects in the memory one at a time, and it abstracts the information so as to only match the most meaningful features.

In this thesis, we will present a framework for recognizing three-dimensional (3-D) objects without resorting to a linear search of all the models in the database — the system's visual memory. We will extend a framework for indexing and matching that will allow us to map an abstraction of the *hierarchical structure* of parts of an object onto a graph, and to utilize a low-dimensional encoding of the graph's structure to quickly prune the model database.

As an example of a decomposition of an object into parts, let us consider the *hierarchical relationships* among the parts of the silhouette of a person's body: four relatively long extremities (arms and legs) and a blob-like shape (the head) subordinated to a trunk. In order to quickly discard non-body like shapes, the idea is to only select shapes with body-like configuration of parts for further analysis. A hierarchy of parts can also

be driven by the level of detail that the part represents, e.g., the fingers can be regarded as the details of a hand-like shape. In turn, dividing the information into parts can allow us to determine whether a person wearing a hat is still similar to our abstraction of a person's body by identifying the hat as an *accessory* part of our model.

Representing Views of Objects

In order to define a 3-D object recognition framework, we first need to decide on the most convenient way of representing all the views in which the object models are likely to be found. Previous research in this area has taken either a 3-D model-based approach, where complete 3-D models are stored in a model database, or a 2-D view-based approach, in which all the expected 2-D model views are stored in the database without explicitly representing the 3-D structure of the objects.

It can be argued that these approaches are, in fact, complementary and that a general-purpose system would require both methods combined. On one hand, 3-D model-based approaches need to search for the model and the geometrical transformation of it that best explains the image. The view-based approach is simpler and does not require the transformation; however, the resulting database is larger, adding to search complexity. The advantage of the model-based approach is that in addition to a compact, object-centered model database, having complete information of the models supports the recognition of unseen views. This feature depends, however, on a 3-D model acquisition process that limits the applicability of the approach. On the other hand, view-based methods do not require complete descriptions of the models; furthermore, they provide a means for dealing with painted and textured objects, for which pixel intensity statistics can also be considered.

A related problem is the representation required to encode the relevant object features and their interrelations. A common strategy in both model-based approaches and view-based approaches has been to decompose the object's shape into a small number of (3-D

or 2-D) primitives forming the building blocks of all possible shapes. Unfortunately, there is no general agreement on what constitutes the primitive parts of all shapes, and whether there is a unique and small set of parts suitable for all objects. Several view-based approaches, such as [30, 29, 43], look for characteristic portions or patches on the object view. The characteristic patches can be those that contain corners, edges, etc. However, this strategy is not sufficient to answer the more general question of how to generalize object properties, which might require the construction of a 3-D model of the object.

Choosing the Right Features

What object features should we use to accomplish 3-D object recognition? There is no concise answer to this question in the literature to date, where the answer given will depend on the kind of objects that we want to recognize, and on whether we want to identify particular instances of objects or any object belonging to a certain class. That is, there are some objects that have a well-defined shape and set of parts, but there are others that are best characterized by their texture (water, trees), colour (fresh food vs. rotten food), painted patterns (books, CD cases), or functionality (chairs, tables, cars). In turn, we also need to agree on whether we want to recognize, say, our dog or any dog?

In this thesis, we will focus on objects that can be characterized by their shape. Different psychophysical experiments support the idea that a large number of objects fall into this category. Pioneering work on this subject was conducted by Biederman [5, 7], who surveys the field in [6]. As an example of these types of experiments, we can consider the work by Hayward et al. [26], in which the authors tested whether people would find contour information useful for recognizing a given set of *unknown objects* from novel viewpoints. They presented subjects with an initial stimulus of a shaded image, and then a second stimulus of shaded images and silhouettes of the object. They found that the recognition was faster for shaded images over silhouettes when the object was

in the same viewpoint, but that there was no difference in recognition performance when the object rotated relatively to the first. The results suggest that shading information is not critical to identifying the objects under viewpoint changes.

Dealing with a Large Database of Objects

Regardless of the features we use, we also need to address the problem of how to *efficiently* handle a potentially large database of object models. A common approach in the database community to deal with the problem of large databases is to create indices that sort the data according to some property. Similarly, in the case of shape matching, we can decompose the problem into an indexing stage and a matching stage. When indexing, the number of objects in the database that might account for the query is reduced by some coarse features; next, in the matching (or verification) stage, a greater number of features in the query are matched to those of the models, yielding a set of feature correspondences and an overall similarity measure for each model.

1.1 View-based 3-D object recognition using shape information

In the present thesis, the aim is to develop a framework to efficiently recognize a potentially large number of objects over changes in viewpoint. To this end, we will use shock graphs as our shape representation, whose topology will form the indexing feature used to prune the model database. A shock graph is a shape abstraction that decomposes a shape into a set of hierarchically organized primitive parts [55]. We will propose a novel shock graph computation algorithm that yields stable graphs under noise and shape deformation due to viewpoint changes. This topological stability of the representation will allow us to combine shock graphs with an indexing and matching framework for hierarchical structures originally introduced by Shokoufandeh et al. in [54]. The

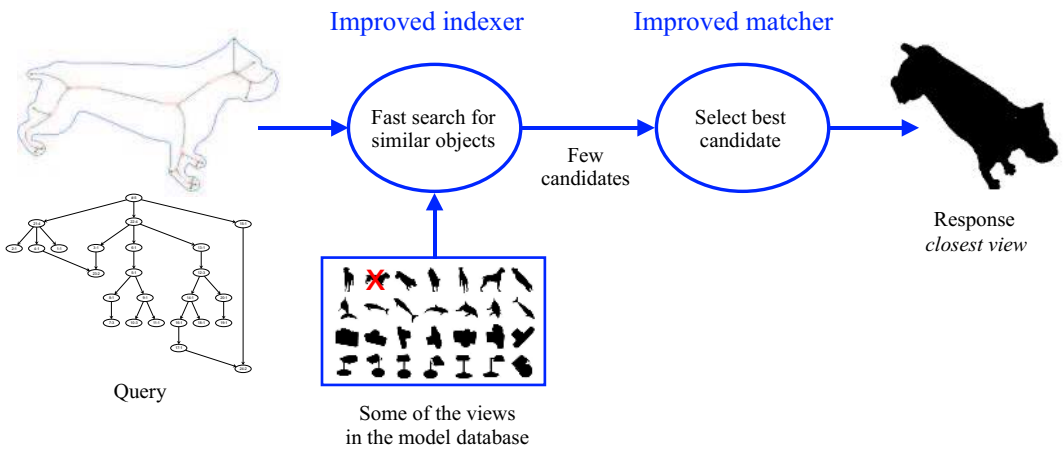


Figure 1.1: The views of 3-D objects are abstracted as graphs of hierarchically organized primitive parts. In our experiments, each of the views in the model database is pulled out from the database and used as the query. The topology of the query graph is used to quickly prune the database by selecting only a few candidates for verification. Next, the matcher computes the similarity between the query and each of the candidates, and sorts them accordingly. The best ranked candidate is considered to be the most similar view to the query among those in the database.

performance of the indexing component of the framework will be improved primarily by an efficient vote accumulation algorithm that finds an optimal solution to the problem of multiple one-to-one assignment of votes by using fixed-size buffers bounded by the number of nodes in the query graph. We will next extend the matching algorithm by ensuring the satisfaction of all the hierarchical constraints embedded in the graphs and by propagating the information contained in intermediate matching results. An outline of the interaction of all these components is depicted in Figure 1.1. Finally, it will be shown that this framework is effective in recognizing object shapes and in estimating the pose of their corresponding 3-D object models. We will demonstrate the performance of the framework with a database of 21 objects and 128 views per object, giving a total of 2688 object views.

Qualitative shape information reduces the number of views per object that need to be stored, and provides flexibility to accomplish generic object recognition. However, using shape information only may not be suitable for various domains in which shape information is difficult to obtain or is not distinctive (see [32] for a discussion on the selection of good features for a given context). For this reason, we will not only explore the solution of using 2-D shape silhouettes for 3-D object recognition, but will also evaluate and extend an indexing and matching framework that provides a general method to deal with other hierarchical representations based on features that may be more appropriate for a given domain.

1.2 Related Work

The concepts regarding each of the components of the thesis, namely the shape representation, the indexing strategy, and the matching algorithm are, to a certain extent, independent from each other. Thus, we prefer to include related work sections in the chapters where each of the components are introduced. Nevertheless, there is some work related to the thesis as a whole. Several authors have looked at the problem of indexing and matching a large number of 2-D views of 3-D objects using shape information.

In [4], Beis and Lowe perform 3-D object recognition from shape information using straight edges as the shape primitives, and group them together according to collinearity and parallelism criteria. The angles and length ratios from these groups are used to form feature vectors, which are indexed from a point database using K-D-trees — a nearest-neighbour search technique. The nearest neighbours to the query vector are weighted according to probability distributions learned in a training stage, and are used to generate hypotheses. The models that receive the most votes are verified using a method for fitting parameterized 3-D models, described in [22].

Similar to the approach followed in this thesis, Sebastian et al. [50] use shock graphs to

represent 2-D shape silhouettes sampled from the object's viewing sphere. The authors, however, propose a hierarchical partitioning of the database, in which shapes are grouped into categories. A small number of exemplars from each category are chosen to represent the grouping, thus forming a database of prototypes used to index into the larger model database. A problem with this approach is that it is not always possible to find shape clusters large-enough to significantly reduce the size of the database. In addition, each match at the prototype level implies a linear search among the shapes belonging to that category, which should be as small as possible.

In recent work, Cyr and Kimia [14] explore the problem of how to partition the view sphere of a 3-D object using a collection of shock graphs. However, they do not address the shock graph indexing problem, and resort instead to a linear search of all views in the database in order to recognize an object.

1.3 Overview

We present a framework for view-based 3-D object recognition in which a query image is represented as a shock graph, a directed acyclic graph (DAG) encoding the hierarchical decomposition of a 2-D shape silhouette into primitive parts. Such primitives arise from a labelling of the singularities along the medial axis of the shape [8]. Shock graphs are presented in detail in Chapter 2, along with a novel algorithm to compute them that improves the topological stability of the graphs under small deformation of the shape boundary due to noise or viewpoint changes.

Given a database of shock graphs representing views of 3-D objects, the shock-graph indexing problem will be formulated as the selection of a small number of candidate object views from the database that might account for the query. Chapter 3 reviews related work on graph indexing and introduces the indexing strategy, proposed in [52], identifying its limitations and exploring a number of extensions to overcome them. In particular, an

efficient vote accumulation algorithm is developed in order to find an optimal solution to the problem of multiple one-to-one assignment of votes.

In the verification step, each of the candidate shock graphs is matched to the query, yielding a similarity measure that is used to rank the candidates. The matching problem is posed as a best assignment problem, in which the node correspondences must satisfy a set of constraints. For the particular case of hierarchical structures, we search for the set of node correspondences that maximizes the shape similarity and also preserves all hierarchical constraints. Chapter 4 discusses the matching algorithm, introduced in [52], and proposes a number of extensions to this algorithm to guarantee that no hierarchical constraint is violated and to avoid other types of incorrect node correspondences.

In Chapters 3 and 4, a special effort is made to present the contributions independently of shock graphs, in order to enhance their applicability to other domains. Chapter 5, in contrast, brings together the different parts of the thesis and evaluates the approach, as a whole, for the specific task of view-based 3-D object recognition. To evaluate the framework, a number of experiments are carried out on large databases of object views by systematically pulling out views from the database, and using them as the query. In addition, we repeat the same experiments, varying the sampling resolution of the views and the degree of missing data in the query.

Finally, Chapter 6 reviews the results and outlines the directions for future research. A number of ideas will be considered on improving the discriminatory power of the indexing mechanism, and on view clustering to further reduce the number of views per object that need to be stored in the model database.

1.4 Summary

We propose a framework in which the silhouettes of 2-D object views are decomposed into hierarchical relationships among primitives. These primitives and their relationships are

represented as DAGs and used for indexing and matching. The general indexing strategy relies on a hierarchical shape description from which a stable structure under visual transformations can be extracted. Topological indices are formed for each graph by a low-dimensional encoding of its structure that allows us to only retrieve *topologically similar* graphs from the model database. In the verification step, a matching algorithm uses these indices, coupled with a domain-dependent node similarity function, to determine node correspondences and rank-order the candidates.

Chapter 2

Shape Skeletons and Shock Graphs

2.1 Introduction

In this chapter, we will briefly review Blum's shape skeleton, along with recent algorithms to compute them which are less sensitive to noise. Next, we will discuss the shock graph, a shape abstraction that groups skeleton points according to the local variation of a radius function. Such a grouping will provide a decomposition of a skeleton into parts, whose interrelation will conform to a well-defined grammar. Finally, we will propose a shock graph computation algorithm and a node similarity function that will satisfy a set of stability criteria.

2.2 The Medial Axis Transform

The skeleton of a shape aims to capture the shape's part structure. One of the first formal definitions of the skeleton is that of Blum [8], who defined the medial axis of a shape as the locus of center points of all the maximal circles contained within the shape boundary. Blum called the shape's skeleton the "Medial Axis Transform" (MAT), an example of which is shown in Figure 2.1.

Blum also gave an alternate definition in terms of a grassfire front that has become

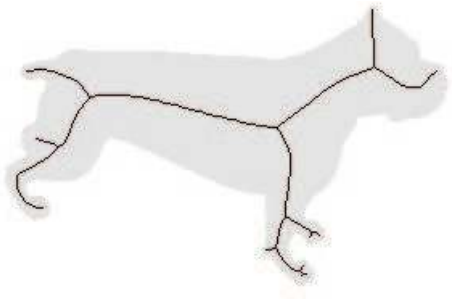


Figure 2.1: Skeleton of a 3-D object's view.

very popular. The idea is to imagine that the shape is uniformly covered with grass and that fire is set simultaneously to all boundary points. The fire front will start propagating toward the center and will extinguish when meeting an opposing front. The collision points of the fronts form the medial axis of the shape. The distance of any skeleton point to its closest boundary points defines the time of formation of the points, and it corresponds to the radius of the maximal inscribed circle.

In order to reconstruct the original shape from the skeleton points, we need the information provided by the radius function, $R(s)$, so as to map every skeleton point, s, to the radius of the maximal inscribed circle centred at s. The original shape can be reconstructed by the union of all disks of radius $R(s)$ centred at s[1].

The first algorithms that were proposed to compute the MAT have great difficulties in the discrete domain of images, and tend to create skeletons that are extremely sensitive to noise and to small deformations along the shape boundary. In order to robustly compute the MAT, a number of researchers, including Blum himself, proposed a variety of solutions. Thus, we can find Blum's Generalized MAT [8], which considers only skeleton points whose radius function is greater than some r. A number of other approaches based on branch pruning and multiscale skeletons were also proposed [21, 45, 44], most of them suggesting a rather ad-hoc solution to the problem.

[1]Note that s can be defined as a function of the Euclidean coordinates x and y in the plane.

In [34], Kimia et al. noticed that the MAT is a special case of a well-studied curve evolution equation from singularity theory, in which the entropy-satisfying singularities, or *shocks*[2], correspond to the skeleton points. More formally, the evolution in time of a curve or wave front, C, can be described by the following partial differential equation:

$$\frac{\partial \mathcal{C}}{\partial t} = F\mathcal{N}, \tag{2.1}$$

where N is the unit inward normal at each point s at time t, i.e., $N(s,t)$, and $F = F(s)$ is the speed of the front at each point. When $F > 0$ is a constant, and the curve at time zero, $C(s,0)$, is set to the initial boundary of the shape, the singular points of the equation correspond to that of Blum's skeleton[3].

This result gave rise to several skeletonization algorithms based on detecting the singular points of Equation 2.1. Several of these new approaches rely on a threshold to determine whether a given point is singular or not [40, 1, 38, 23]. It turns out that there seems to be no such threshold that will ensure a one-pixel-thin connected skeleton for which the union of maximal disks will exactly reconstruct the original shape. In [15], Dimitrov et al. solve this problem by setting a conservative threshold, and then applying a thinning algorithm that simultaneously preserves the shock connectivity and its homotopicity with the original shape. The latter algorithm succeeds in coping with small noise along the shape boundary. All the shape skeletons for the figures in this chapter have been computed with this algorithm.

[2]A shock, or shock point, is the usual term that refers to a skeleton point when the algorithm used to compute the skeleton is based on Equation 2.1. Nonetheless, it is harmless to consider this term as a synonym of skeleton point. Other terms that have become synonyms are: MAT, (Blum's) skeleton, and grassfire transform, and also time of formation as referring to the value of a shock's radius function.

[3]In singularity theory, F is set as a function of the curvature at each boundary point: $F = (1 + \alpha\kappa)$, where $\alpha \geq 0$ is a regularizing parameter and κ is the curvature. Hence, for $\alpha = 0$ we obtain the MAT of the shape [34].

2.3 Representing Shapes

The skeleton is a transformation of the shape that highlights the symmetries of the boundary. Hence, it does not provide an abstraction of the shape, but instead, it can be used to facilitate the task of computing such an abstraction. There are a large number of shape representations that do not require computing the shape skeleton. The main argument to perform this transformation is that the skeleton simplifies the decomposition of a shape into parts which, in turn, provides a means for dealing with partial occlusion and articulation of parts. For our particular task, we are interested in *hierarchical* representations that satisfy Marr's stability requirement [41]: local changes to the shape must result in local differences in the representation. Our goal will then be to use the stability of the representation and its hierarchy of elements to efficiently index and match shapes. To this end, we will next study the shock graph: a directed acyclic graph representing the decomposition of a skeleton into primitive parts.

2.3.1 Shock Graphs

Inspired by Blum's original idea of using directed graphs to define equivalence classes of shapes, Siddiqi and Kimia defined the concept of a shock graph [55]. A shock graph is an abstraction of the skeleton of a shape onto a directed acyclic graph (DAG). The skeleton points are first labelled according to the local variation of the radius function at each point. Type 1 shocks form a *segment* of skeleton points, in which the radius function varies monotonically, as is the case for a protrusion. A type 2 arises at a neck, and is immediately followed by two type-1 branches flowing away from it in opposite directions. Type 3 shocks belong to an *interval* of skeleton points, in which the radius function is constant. Finally, a type 4 arises when the radius function achieves a strict local maximum. That is, the boundary collapses to a single point. An example is shown in Figure 2.2. Next, we will provide a formal definition of a shock graph based on that

in [56].

The construction of the graph can be divided into three steps, each of which requires a definition. To this end, we will consider the set of skeleton points S to be a continuous and connected set of the medial axis points of a closed curve, as defined by Blum [8]. Two points in a set are said to be connected iff they can be joined by a continuous path of points also in the set. In order to present the definitions based on a continuous set of skeleton points, as it was originally done in [56], we need to translate the concept of point adjacency from the discrete domain to the continuous domain. To this end, let $N(s, \epsilon)$ be the set of shocks in $S \setminus \{s\}$ within distance ϵ from s, and let $N(s)$ be the set of the *largest connected sets* of points in $S \setminus \{s\}$. That is, respectively, the set of skeleton points that do not include s, and the set of connected components originated by removing s from the skeleton.

First, we will define the labelling of each point in a *skeleton branch* according to its time of formation. A skeleton branch is formed by all the connected medial axis points between two endpoints. An endpoint is a skeleton point s s.t. $|N(s)| \neq 2$, i.e., terminal points and junction points[4]. Next, we will determine how the labelled points in a given branch are to be grouped according to their labels and connectivity, so that each group of same-label connected points will be stored in a graph node. One skeleton branch will give rise to one or more nodes. Finally, we will specify how to add edges between the nodes so as to produce a directed acyclic graph with edges directed according to the time of formation of the shock points in each node.

Definition 1 (labelling) Let $R(s)$ be the radius function $R : S \mapsto \mathbb{R}^+$, and $l(s)$ be a shock labelling function $l : S \mapsto \{1, 2, 3, 4\}$, where S is the set of *continuous* medial axis points (i.e., shocks) of shape X. Then $l(s)$ for $s \in S$ is defined as:

[4]Junction (or bifurcation) points are considered to belong to all the branches they are connected to, and so repetition of junction points will be allowed in the shock graph nodes.

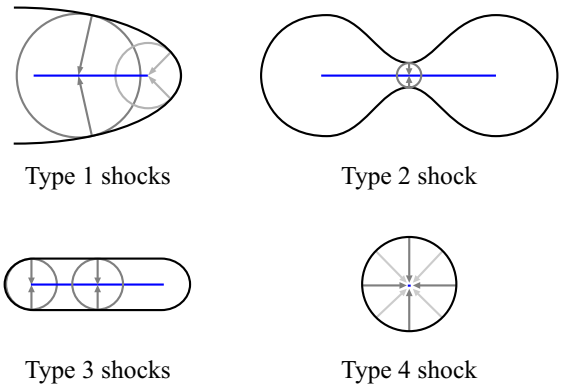


Figure 2.2: Skeleton points labelled according to time of formation.

$$l(s) = \begin{cases} 4 & \text{if } \exists \epsilon > 0 \text{ s.t. } R(s) > R(s') \ \forall s' \in N(s,\epsilon), \\ 3 & \text{if } \exists \epsilon > 0 \text{ s.t. } R(s) = R(s') \ \forall s' \in N(s,\epsilon) \neq \emptyset, \\ 2 & \text{if } \exists \epsilon > 0 \text{ s.t. } R(s) < R(s') \ \forall s' \in N(s,\epsilon) \text{ and } N(s,\epsilon) \neq \emptyset \\ & \text{and } N(s,\epsilon) \text{ is not connected}, \\ 1 & \text{otherwise.} \end{cases} \tag{2.2}$$

Alternatively, this definition can be regarded as specifying how to label the skeleton points according to their velocity dR/ds, and their acceleration d^2R/ds^2. To label a skeleton point, s, we only need to look at the signs of its instantaneous velocity and acceleration so as to determine whether $R(s)$ is: monotonically increasing/decreasing (type 1), a strict local minimum (type 2), constant (type 3), or a strict local maximum (type 4).

Definition 2 (grouping) Let $B_1 \ldots B_n$ be the *largest* groups of *connected* shocks in S s.t. $\forall s, s' \in B_i$, $l(s) = l(s')$ and $\forall s \in B_i$, either $|N(s)| \leq 2$ or if $|N(s)| > 2$, then s must be a terminal point of B_i (i.e., $B_i \setminus \{s\}$ is connected), for $1 \leq i \leq n$. Let the group's label, $l(B_i)$, be the label of the shocks in B_i, and similarly, let the time of formation of the group, $t(B_i)$, be the interval $[\min_s(R(s)), \max_s(R(s))]$ defined by the time of formation $R(s)$ of all $s \in B_i$.

In other words, the shock groups are skeleton segments in which all the shocks have the same label and belong to the same branch. Moreover, note that each bifurcation point will belong to more than one group if the point's label is the same than those of the groups of points meeting at the bifurcation.

Definition 3 (Shock Graph) The shock graph of a 2-D shape is a labelled graph $G = (V, E, \gamma)$ such that:

vertices $V = \{0, \ldots, n\}$, corresponding to the groups $B_1 \ldots B_n$, and 0 denoting a root node;

edges $(i, j) \in E \subseteq V \times V$ directed from vertex i to vertex j if and only if $i \neq j$, and $i \neq 0 \wedge t(B_i) \geq t(B_j)$[5] $\wedge$ $B_i \cup B_j$ is a connected set of shocks, or $i = 0 \wedge \forall k \neq 0\ (k, j) \notin E$;

labels $\gamma : V \mapsto \{\#, 1, 2, 3, 4\}$ s.t. $\gamma(i) = \#$ if $i = 0$ and $\gamma = l(B_i)$ otherwise.

From the above definition, it can be seen that the shock branches act as the shape's primitive parts which, in turn, are represented as nodes in the graph. The edges are directed according to the relative time of formation of each part such that the last shocks to form are at the top of the hierarchy of dependencies. Thus, the central parts of the shape will be at the top, and the smaller shape extremities will be further down in the hierarchy.

2.3.2 The Shock Graph Grammar

One of the most appealing characteristics of shock graphs is that any shock graph can be described by a grammar composed of an alphabet of five symbols: {# (the start symbol), 1, 2, 3, 4, Φ (the terminal symbol)}, and ten rewrite rules: see Figure 2.3. The grammar constrains the possible arrangement of primitive parts that form shapes

[5]The binary relation $\geq$ for the intervals $t(B)$ can be defined in different ways. In this thesis, we define it as $t(B_i) \geq t(B_j) \Leftrightarrow (\max_s(R_i(s)) > \max_s(R_j(s))) \vee (\max_s(R_i(s)) = \max_s(R_j(s)) \wedge \min_s(R_i(s)) \geq \min_s(R_j(s)))$.

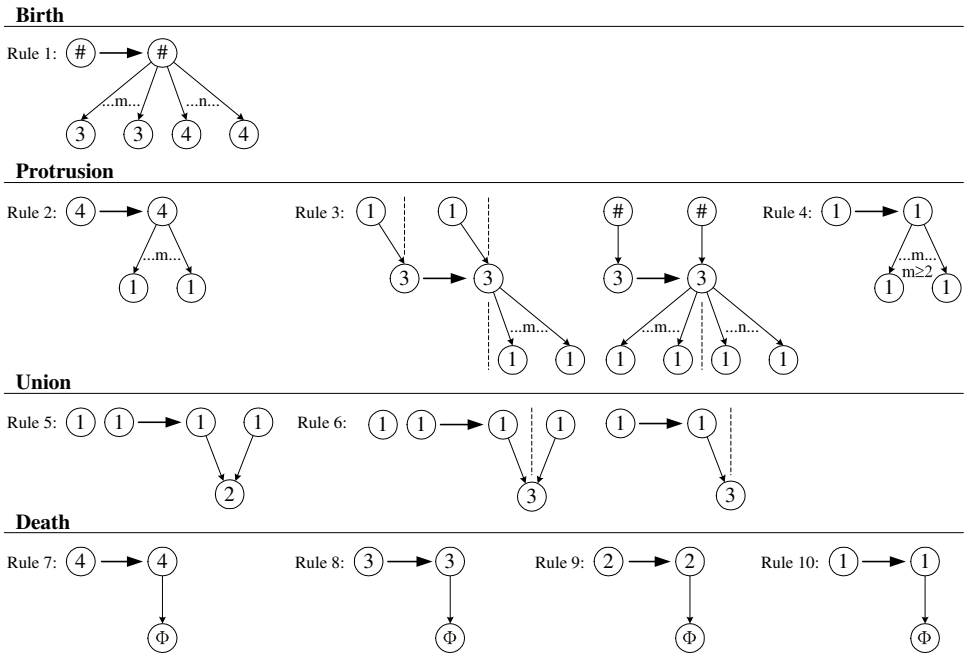


Figure 2.3: The Shock Graph Grammar. Please refer to [56] for details.

and, furthermore, it allows for the identification of semantic equivalences among parts. The rows in 2.3 group the sequence of symbols that are semantically equivalent into four types: birth, protrusion, union, and death.

2.3.3 Limitations and Possible Solutions

The labelling of skeleton points or shocks is invariant to translation and rotation of the shape. In addition, since the labelling is locally determined, it is stable under uniform scaling and within-class part deformation. However, we can notice some limitations of the representation when considering skeletons computed from discrete images. On one hand, scale and viewpoint changes often originate new branches in a skeleton due to small details on the boundary that were not identified at a lower scale or that were not visible from a different viewpoint. These new branches can affect, to a certain extent, the structure a shock graph. On the other hand, the shock labels depend on the pairwise difference of

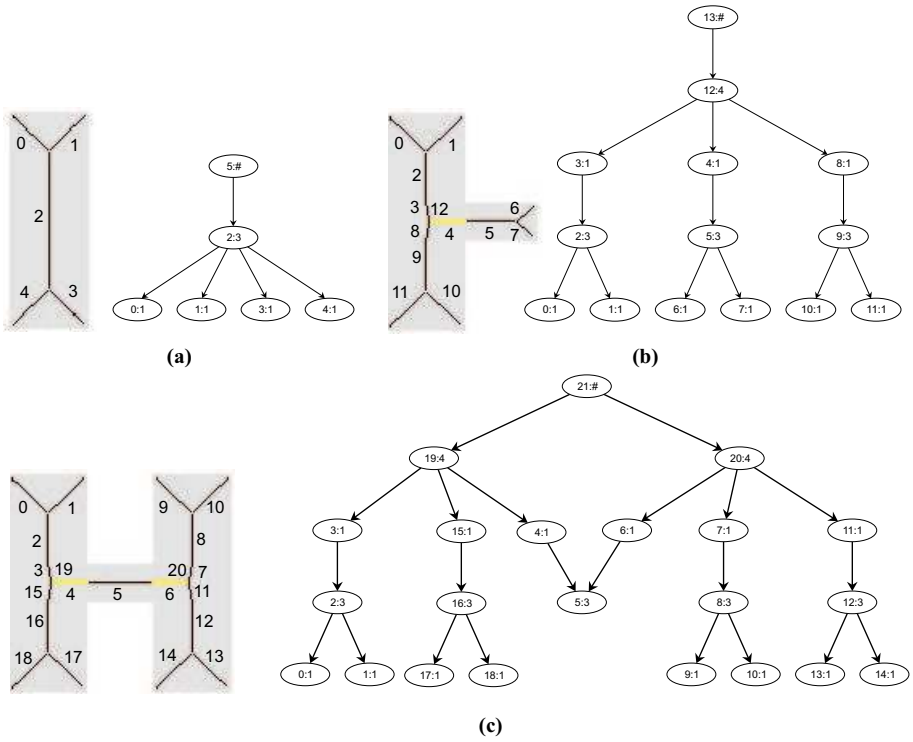


Figure 2.4: Skeletons and their shock graphs. Skeleton segments and their corresponding nodes are labelled with the same ID. The displayed attributes of each node are ID:LABEL. (a) A simple shape. (b) A new *part* is added and linked by a ligature segment (yellow). (c) A third component is added; segment 5 acts as a neck in the new context.

the radius function at adjacent shocks (see Eq. 2.2), which requires, in practice, to define some threshold to increase the tolerance to differences in radii. For example, we could redefine the equality condition for type 3 in Equation 2.2 as, $R(s) = R(s') \pm \tau$. However, the choice of τ depends on the scale and so, to output the same results at different scales, τ must be defined as a function of the relative scale of the given skeleton segment. We will come back to the later limitation in Section 2.5.1, in which we will propose a robust method for labelling the skeleton points that will also accommodate small differences in scale.

Another source of instability in shock graphs has been studied by August et al. [2]. They have shown that concave corners in the shape boundary produce skeleton segments

where all the points are related *only* to the corner points at the boundary. Thus, small changes in the positions of these corner points will cause rather big changes in the *ligature segments*, which will ultimately give rise to differences in the corresponding shock graph (see Fig. 2.4). In [2], the authors propose to eliminate the nodes in the shock graphs caused by ligature segments and show how this simple strategy produces more stable graphs.

Finally, on a rather implementation-related aspect, we can notice that the shock labelling function, 2.2, defines type 2 and type 4 nodes as single points. Thus, small noise or small variations in shape can turn these nodes into a type 3. Such label variation is, in fact, captured in the grammar, which specifies that a type 3 acting as a neck is semantically equivalent to a type 2 (Rule 6), and a type 3 in a birth process is equivalent to a type 4 (Rule 1). This problem can be solved by either labelling type 3 nodes as 2 or 4 according to their local context, or by factoring in the local context of a node into the node similarity function (next section).

2.4 Node Similarity Functions for Shock Graphs

Shock graphs define meaningful equivalence classes for shapes: two shapes abstracted by the same graph are expected to be perceptually similar at a coarse level. This property will allow us to index a model database to recover a set of candidate models with similar shock graphs. The next step will then be to verify each of these candidates by evaluating the information that we have ignored in the abstraction. With this goal in mind, a list of attribute vectors is associated with every point s in every node in the graph. Such vectors are 4-tuples (x, y, t, α) representing the attributes of each shock point in the node: Euclidean coordinates (x, y), time of formation t, and direction α. The set of attributes of each node are used to define a local measure of similarity between two nodes in different graphs.

Several authors have proposed different definitions of node similarity functions for shock graphs. For instance, Pelillo et al. [46] compute a weighted sum of each of the four components of all the attribute vectors in a given node. The terms in such a summation correspond to the length difference and the average difference in: radius, velocity and direction. Since the two segments being compared are likely to differ in the number of shocks, the m shocks in the longest segment are mapped to the n shocks in the other by $\xi(i) = \lceil \frac{in}{m} \rceil$. Clearly, such a mapping of shocks may not be the one that best represents the similarity of the segments. In contrast, Klein et al. [35] propose a dynamic programming algorithm to find the best mapping that will maximize a comparable similarity function[6]. In [56], Siddiqi et al. suggest a different approach to the problem. The idea is to choose five equidistant points in each segment and then compute the affine transformation that best aligns the 4-D curves. The overall *dissimilarity* between both curves is considered to be the Hausdorff distance among *all* the 4-D points in the aligned curves.

To the best of our knowledge, there has not been a formal comparison among the above similarity functions. A brief examination of them, however, will reveal that finding the best mapping among all the points in the curves may provide the best solution as long as scaling transformations can be ignored. Unfortunately, this algorithm has time complexity $O(|G_1|^2\ |G_2|\ \text{diameter}(G_1)\ \text{depth}(G_2)^2)$, and so the simple approximation of Pelillo et al. may be a better choice for certain applications. When scaling transformations are to be accounted for, none of the first two will do, and therefore computing the best alignment of the 4-D curves becomes a reasonable alternative.

2.5 Computing Stable Shock Graphs

Recall the definition of shock graph labelling introduced by Siddiqi et al. [56] that is presented in Section 2.3.1. In the definition, the shock labels are determined by the

[6]More precisely, they define a measure of dissimilarity, or cost, in which the attributes' differences are not, in fact, averaged or weighted.

radius function at each point's local neighbourhood. Unfortunately, in the case of discrete skeletons, the location of shock points is very sensitive to the accuracy with which both the radii and centers of the locus of the maximal inscribed circles can be computed. Clearly, perturbations to the shape's boundary will affect the exact pixel location of the shock, regardless of the algorithm used to compute the skeleton. Furthermore, the skeletonization algorithms proposed to date also suffer from instabilities, due to the discrete nature of images. In [15], for instance, Dimitrov et al. propose a dynamic programming algorithm to find the locations of skeleton points in neighbourhoods defined by the singular values of equation 2.1. This method guarantees a connected skeleton, but cannot avoid small fluctuations in the locations of the shock points within each neighbourhood.

2.5.1 Robust Labelling

Our problem is to find a labelling of the skeleton points that is robust to small perturbation of the boundary and to noise in the location of the points. A first attempt to ameliorate these problems would be to smooth the radius function by some weighted average of each point's neighbourhood. Unfortunately, this solution will not eliminate the problem completely and at the same time, will introduce new artifacts that will lead to a new source of instabilities.

We propose instead a model selection approach to the problem, which can naturally accommodate the instabilities of the radius function. Our method consists of finding a small number of line segments to approximate the radius function for each skeleton branch. Before we develop the algorithm, recall that we can think of the labelling as a function of the instantaneous velocity, dR/ds, and acceleration, d^2R/ds^2, at each point. Thus, in order to compute the derivatives of $R(s)$, we would first need to apply a smoothing filter to the data to obtain a differentiable function. However, the signs of the derivatives are highly sensitive to noise in the data, and so we will follow an alternate approach.

We will approximate the data with line segments, whose slopes will give us the velocities of the points that they model. The line slopes will allow us to label segments of points as either type 1 or type 3 shocks. Next, by considering the shock graph grammar, we will infer the type 2 and type 4 points in a robust fashion.

Fitting the data with line segments instead of using segments of a higher-degree polynomial may seem to be, at first, very limiting since it is easy to imagine, for example, a shape part for which its radius function will be best approximated by a quadratic function. It is rare, however, to encounter a piece of a skeleton branch whose radius function requires higher degree polynomial segments to accurately fit it using fewer segments. In our experiments, we have observed that most radius functions have piecewise linear behaviour and only a few have segments with quadratic behaviour. Hence, in principle, by using line segments to approximate a radius function, we will be forced to use extra segments when the radii of the skeleton points change quadratically. Fortunately, this will turn out not to be a problem for our goal of labelling the points.

We will assume that all the data points modelled by a particular line segment have equal velocity, which is given by the slope of the line. Thus, the labelling problem reduces to examining whether the segment's slope is zero, positive, or negative. Since we are only concerned with the sign change of the velocity, it should be clear that we do not need to capture the exact shape of $R(s)$. Rather, an approximation of $R(s)$ by a number of line segments will suffice. The first step in the construction of shock graphs is then to group adjacent skeleton points with equal sign or zero slope into a graph node, and label the node as type 1 or type 3, respectively.

Model Selection

Our model selection problem is defined as finding a small number of line segments (components) that best describes the radius function for a given skeleton branch. To illustrate the problem, consider the plots of the radius function for the torso and neck of the boxer

in Figure 2.5. The first task is to find the number of line segments that we need, along with their parameters. We will also require these model parameters to be independent of rigid transformations of the shape, which may cause the radius function to be mirrored. One of the main difficulties of the fitting is dealing with points at inflection locations, where the ambiguity in the line segment ownership is more pronounced. The method that we will propose next will find a robust solution to this problem.

We will now turn to the discrete domain of images, and will assume that the skeleton S is a discrete and connected set of points in $\mathbb{N}^2$, with the local neighbourhood of a discrete point taken to be the 8-neighbourhood. We are interested in approximating the radii of skeleton points as a function of the cumulative piecewise linear distance, d_i, along a given skeleton branch with endpoints s_0 and s_n, where $s_i = [x_i, y_i]$, for $0 \leq i \leq n$. This distance is given by $d_i = \sum_{k=0}^{i-1} ||s_{k+1} - s_k||_2$, and the radius $\hat{R}(d_i)$ of the shock point s_i at distance d_i from s_0 is equal to $R(s_i)$.

The error of a given line segment fit is computed by its least-squares error. In our case, we do not expect outliers to occur and so an unweighted least-squares method will provide a good approximation. To compute the $n+1$ indices of endpoints for n line segments that minimise the fitting error, we define the following function:

$$\hat{E}(n,i,k) = \begin{cases} \text{LSF}(i,k) & \text{if } n = 1, \\ \min\limits_{i<j<k} \left\{ \hat{E}(\lfloor n/2 \rfloor, i, j) + \hat{E}(\lceil n/2 \rceil, j, k) \right\} & \text{otherwise;} \end{cases} \tag{2.3}$$

where $\text{LSF}(i,k)$ is the line and its associated error that best fits, in the least-squares sense, the data between endpoints indexed by i and k, i.e., $\text{LSF}(i,k) = e(m_{ik}, b_{ik})$, for $e(m,b) = \sum_{j=i}^{k} (R(d_j) - (md_j + b))^2$ and $(m_{ik}, b_{ik}) = \underset{m,b \in \mathbb{R}}{\text{argmin}} \{e(m,b)\}$. In turn, $\hat{E}(n,i,k)$ is the minimum error that can be achieved when fitting points i to k with n segments. Note that the segments are constrained to be continuous on s but not on $R(s)$.

This function is easy to implement by dynamic programming or memoization techniques, so as to avoid computing the same cases more than once. Then, we can efficiently find the smallest value of n whose minimum error is smaller than a given threshold. That

is, we start with $n = 1$, and increment n by one until the minimum error of fitting the function with n segments is less than a threshold τ_e. This threshold is set so as to consider small variation in wide and short branches to be less significant than that for narrow and long ones. In other words, if we set τ_e as a function of the number of points in the (connected) branch, we can capture the perceptual significance of the error[7].

The choice of endpoints for the fitting segments is sensitive to small variation of the shape's boundary, which can cause the endpoints to fluctuate slightly. However, we will see that the labelling of points will not be affected by the exact choice of endpoints, since a different selection of endpoints is not expected to affect the slopes of the segments to a large extent so as to change the qualitative labelling of the segment.

Figure 2.5 illustrates the results of the above model selection algorithm for different branches of the boxer's skeleton. The algorithm naturally avoids small regions that could be labelled as changes in direction of velocity, but that are, in fact, caused by small perturbations to the boundary or by noise from the skeleton computation. Consider, for example, the challenge of labelling the points of branches 12 and 15. Both branches have very unstable values that do not correspond to meaningful events on the shape boundary. In both cases, the line segments found provide a good representation of the radius function and more importantly, of the contours they encode.

Grouping

Now that we have a *small* number of line segments that approximate each skeleton branch's radius function, the labelling and grouping of shock points into graph nodes becomes easy. Points modelled by adjacent segments with equal slope sign are grouped together in a graph node. The overall algorithm to construct a robust shock graph from

[7]In our experiments, we set this threshold to be 1/2 of the number of shock points in the branch. Note that the number of points in a connected skeleton is proportional to the scale of the shape, and so this method can accommodate small changes in scale. The precise value for the threshold is not significant, provided that it causes the desired effect and is consistent across all models.

the skeleton of a shape can be stated as follows:

1. Given a set of points $\{s_i\} \in \mathbb{N}^2$, and a radius function $R(s_i)$, with $\{s_i\}$ thinned to form a one-pixel-thin connected skeleton. Compute the 2-connected branches.

2. Model the skeleton radius function of each branch with a small number of line segments, as discussed in Section 2.5.1.

3. Group shock points into graph nodes, and label them as either zero velocity (type 3), or positive/negative velocity (type 1), as specified in Section 2.5.1. Add an edge between any two nodes whose endpoints are adjacent in the skeleton (the direction of the edge is not important at this point).

4. For all junctions in which all the nodes meeting at the junction are type 1 and have equal velocity sign w.r.t. the meeting point, add a single-shock node adjacent to all of them, and label it either as type 2, if the velocity sign of the nodes is positive w.r.t. the junction, or as type 4 otherwise.

5. Order the nodes in the graph so as to preserve their adjacency and satisfy the shock graph grammar, while directing the edges from nodes with greater radii to those with smaller radii (Definition 3). This step has been proven to exist and to be unique for any closed curve [56].

6. Add a root node labelled as #. Next, add edges directed from the root node to every other node in the graph with in-degree equal to zero, so as to obtain a single rooted directed acyclic graph.

7. (optional) Relabel every type 3 node with only two adjacent type 1 nodes, one on each side, as either type 2 if the radius functions of the type 1 nodes increase away from the type 3 node, or as type 4 if the radius functions of the two nodes decrease w.r.t. the type 3 node.

The last step of the above algorithm is optional, mainly because it violates the definition of a shock graph, which requires type 2 and type 4 nodes to be composed of only one shock point. However, we have found that this restriction weakens the topological stability of the graph, for small perturbations to the shape will easily transform a one-shock type 2 (a neck) into a type 3 if the neck becomes even one pixel longer. This step will then improve the stability of shock graphs in terms of node labelling, leading to similar shapes being represented by shock graphs with equal or similar topology and node labels.

Close to Zero Velocity

Shock points modelled by line segments with zero slope are labelled as type 3. In practise, however, it is rare to find line segments with exactly zero slope, and so we need to set a threshold that determines when a slope can be considered to be equal to zero. When doing this, we also want to capture the relative significance of the slope in terms of its context. The perceived difference in the values of skeleton point radii is analogous to the speed of a car, in which a change in speed of 15 km/h in, say, 4 seconds is more significant if our current speed is 50 km/h than if it is 180 km/h. We account for this simple notion as follows: given a line segment defined by the line $md + b$ in the interval $[d_0, d_1]$, we set a threshold, τ_z, on the ratio

$$\frac{m(d_1 - d_0)}{m\frac{(d_0+d_1)}{2} + b} \leq \tau_z,$$

which is the ratio of the difference of radius in the interval and the actual radius at the center of the interval. Note that this ratio is always defined since the radius function at every shock point is positive and so $m = 0$ iff $b > 0$.

Skeleton points modelled by a line segment with relative change in the radii at the endpoints below threshold are labelled as type 3. The advantage of a stable identification of zero velocity shocks is twofold. Firstly, a stable labelling of the points will allow us, when indexing and matching, to only consider node correspondences between nodes with

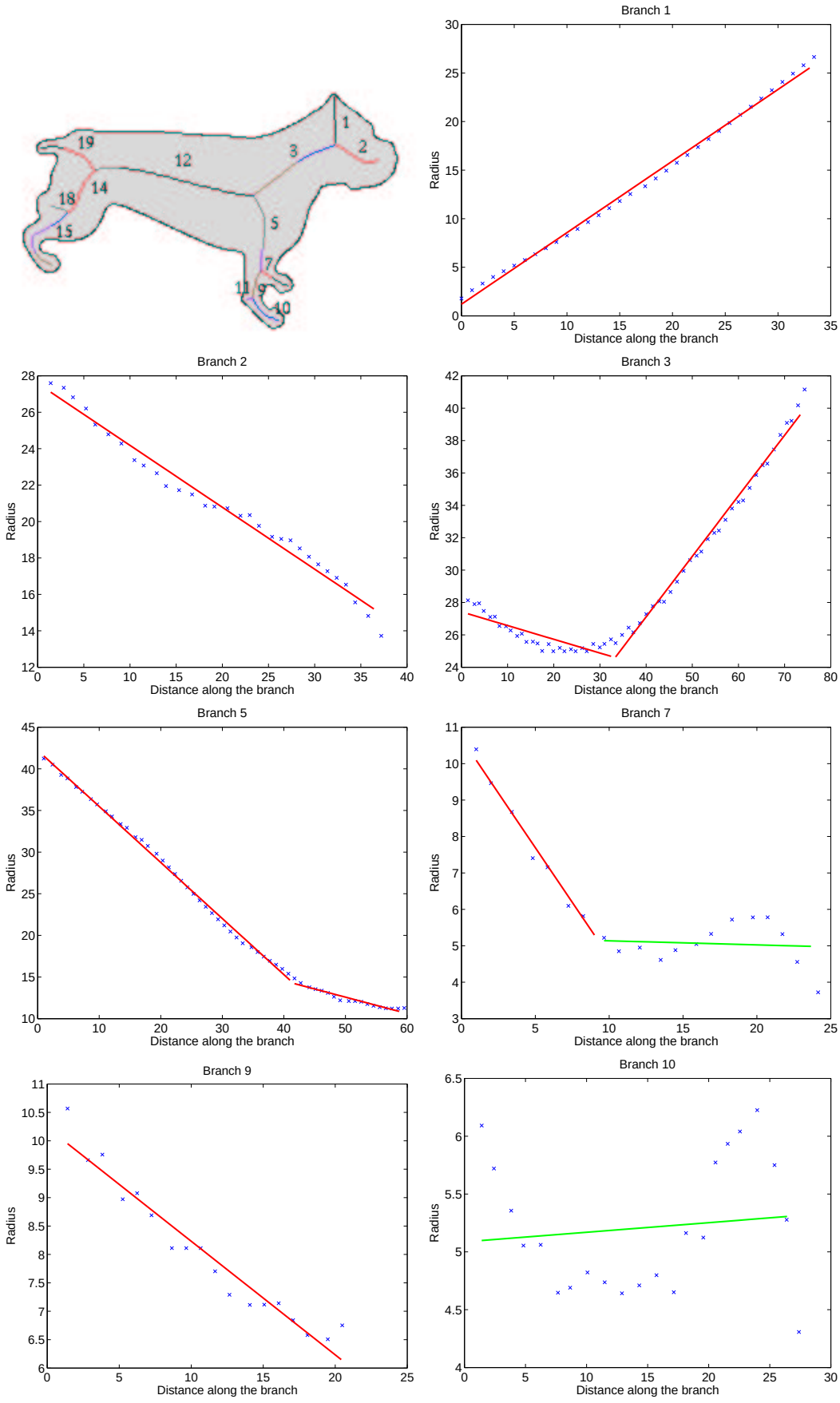


Figure 2.5: Skeleton of a view of the boxer. The plots continue on the next page.

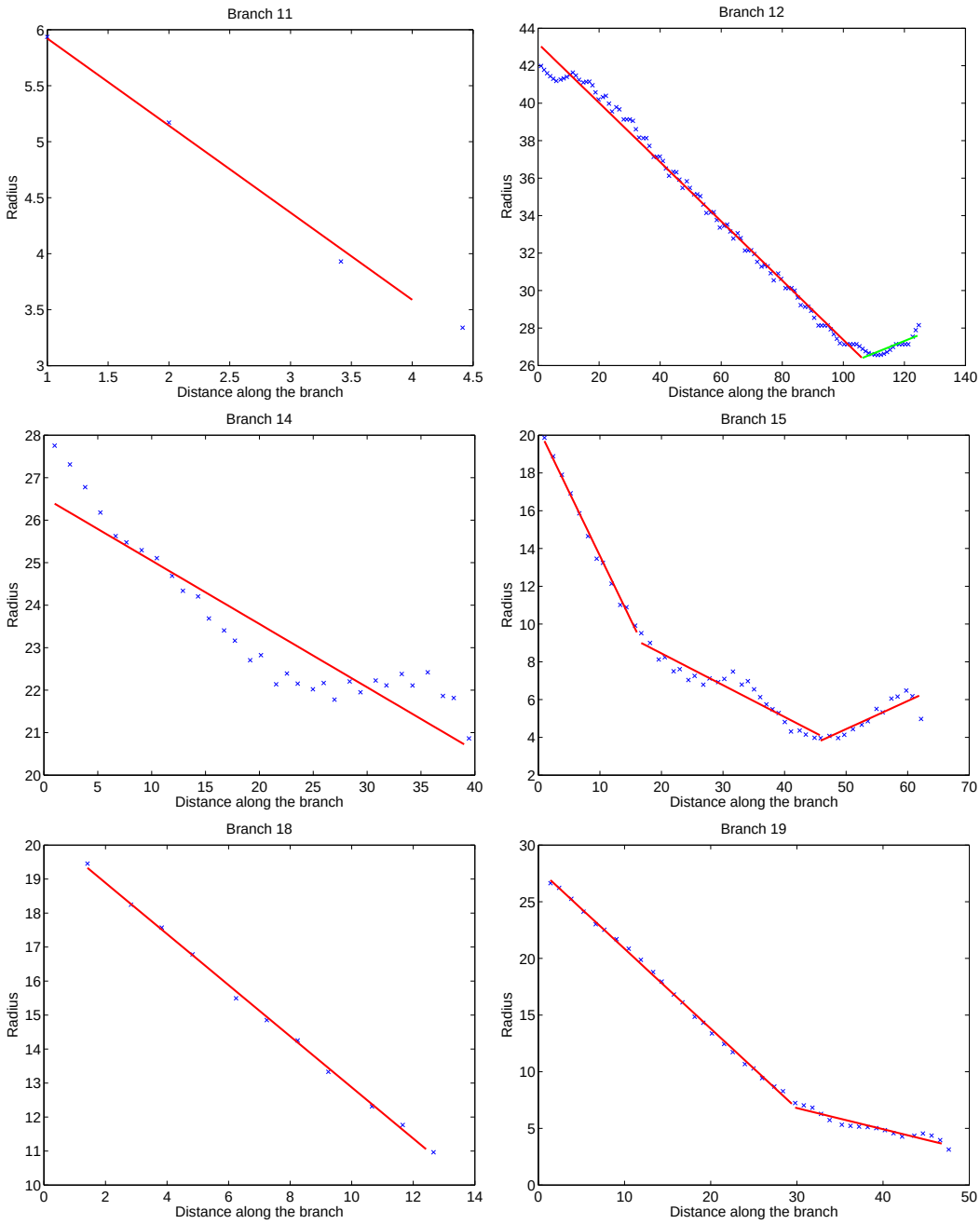


Figure 2.5: Each figure plots the values of the radius function corresponding to a skeleton branch of the boxer. The data points are approximated by *a few* line segments, which are shown overlaid on the points. The colouring of the skeleton shows the resulting grouping of shock points. Note that the scaling factor is different for each plot. Branch 10, for example, is modelled by only one line since the fitting error is 6.71, which is below threshold $\tau_e = 11$, set as half of the number of points to fit. Segments with relative change in radii below threshold τ_z are show in green. Endpoints that are shared by several branches are displayed but not included when fitting.

the same label. Secondly, it can be seen that the labelling of the points affect the structure of the graph, and so a stable labelling leads to a structural stable shock graph.

Figure 2.7 shows an example of the structural changes related to the labelling of the skeleton. In this figure, the slope of segment 15 is greater that τ_z (see also the plot of branch 15 in Figure 2.5), which causes not only that node 15 is labelled as type 1, but also the addition of a type 2 (node 24, a local minimum), and a type 4 (node 25, a local maximum). In contrast, a terminating type 3, such as node 7, does not require the addition of the type 2 and type 4 nodes[8]. Therefore, a labelling of the skeleton points that is not highly sensitive to small changes on the boundary is crucial to obtain structurally stable graphs.

Acceleration and Ligature Segments

We have mentioned that skeleton points modelled by adjacent line segments with equal slope sign are to be grouped together. This step corresponds to the definition of shock graphs, which states that regardless of the velocity or the acceleration, monotonically increasing or decreasing adjacent shock points are to be labelled as type 1. However, we have found that the endpoints of the ligature segments analysed in Section 2.3.3 are related to *significant* accelerations, i.e., to a large difference between the slopes of two adjacent line segments. Ligature segments are of key importance to our task of recognition, because they signal the attachment of shape parts. Figure 2.6 shows the same object from different viewpoints and the corresponding deformation of shape parts in each view due to perspective projection. At matching time, we will compare the attributes of pairs of nodes, and will compute a measure of their similarity to obtain a *one-to-one* node correspondence. Hence, detecting ligature segments and grouping them into distinct nodes will allows us to reduce the impact of the deformations by isolating parts that will not likely be affected in the same way.

[8] A type 3 segment has no strict local minimum or maximum.

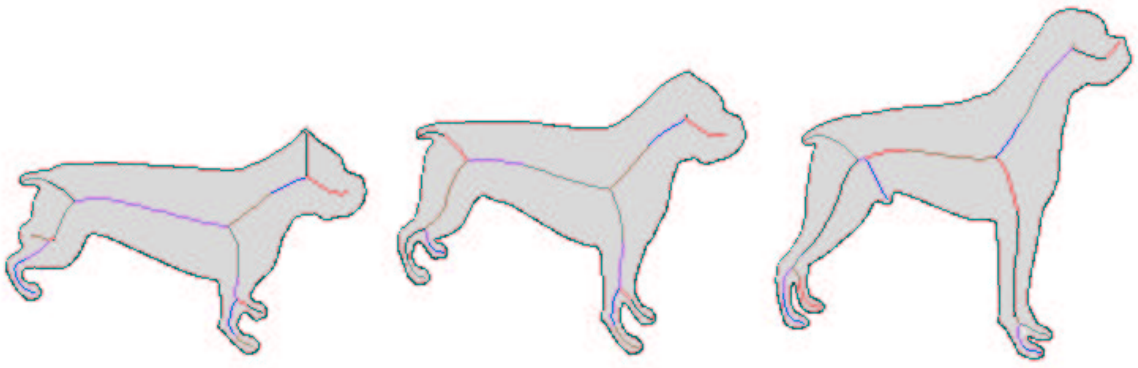


Figure 2.6: Views of the boxer labelled with the algorithm described in Section 2.5.1. Adjacent skeleton points with the same label are coloured similarly, where each group of same-colour adjacent points represents a single node in the shock graph of the shape. Note how the ligature segment that attaches the front leg(s) to the torso undergoes a different deformation than that of the segment corresponding to the leg. The ligature segments in these images were detected by the method described in this section.

It remains only to determine when an acceleration is significant. Once more, we must account for a relative notion of change. To this end, let m_0 and m_1 be the slopes of adjacent line segments with equal sign. Then, we group together the points associated with these segments if

$$\frac{|m_0 - m_1|}{\max(|m_0|, |m_1|)} \leq \tau_l,$$

where τ_l is the ligature segment threshold[9]. The main disadvantage of this method to detect ligature segments is that it is difficult to find a value for τ_l that precisely separates ligature segments from non-ligature segments in all possible cases. We believe that other methods based on detecting skeleton points associated with concave corners[10] on the shape's boundary might provide more stable results. Unfortunately, these methods

[9]Note that $\max(|m_0|, |m_1|) > 0$ because in this step we are dealing only with type 1 nodes.

[10]The definition of a concave corner will also depend on a threshold. However, this concavity threshold might be more naturally set than our τ_l.

require mapping skeleton points to boundary points, and so they are more difficult to implement. Furthermore, we have found that in practice, our approach behaves well for most cases.

It should be noted that by splitting segments of equal slope sign, we are again departing from the strict definition of shock graphs. This operation violates rule 4 of the shock graph grammar (see Section 2.3.2), which states that a type 1 node can only have two or more type 1 nodes as its children. Our (optional) modification relaxes this constraint and allows chains of type 1 nodes to exist. We can also think of this step as an extension to the grammar that adds a new type (5) to label ligature nodes, i.e., nodes whose shock points belong to a ligature segment, which are described in Section 2.3.3.

Figure 2.7 shows the resulting shock graph for one of the views of the boxer. The graph was computed by the algorithm outlined above, including all the optional extensions to the shock graph definition[11]. In this case, nodes 5 and 6 should be one node according to the definition of shock graphs, but are split in two due to the abrupt change in acceleration, shown in Figure 2.5 for branch 5. This is also the case in branches 15 and 19; in all cases, the pronounced acceleration change signals a ligature segment. In addition, note that nodes that are labelled as type 3 in Figure 2.7 are those that are modelled by a line segment with relative change in radii at the endpoints (Figure 2.5) below threshold τ_z.

2.6 A Viewpoint Invariant Node Distance Function

We have shown a method for computing stable graphs in terms of their topology which, as we will see in the following chapters, has an important *global* impact on both the indexer and matcher. We now need to address the problem of node similarity in order to solve the *local* domain-dependent component of the matching process. Our method

[11]The plots of Figure 2.5 show the line segments overlaying only the points they model, i.e., without including the endpoints shared by junction points between segments.

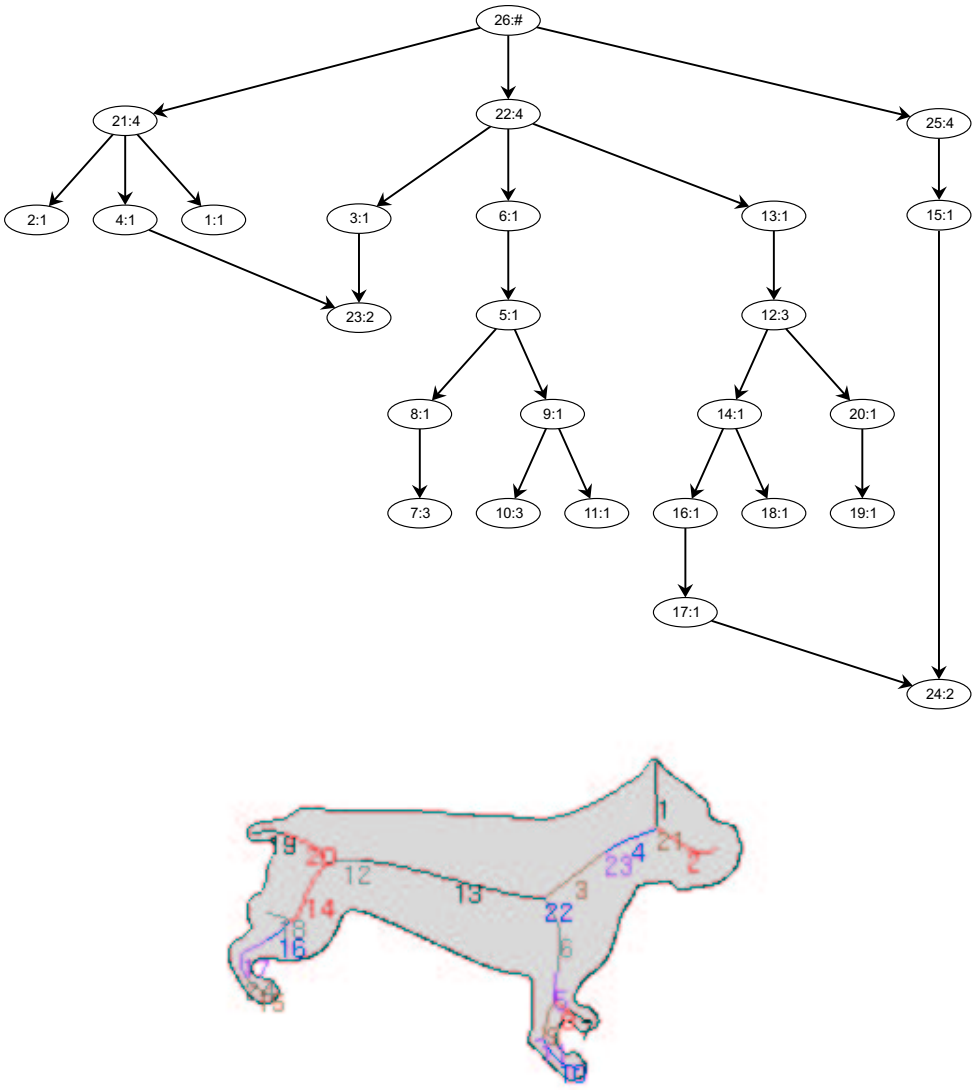


Figure 2.7: The resulting shock graph for the skeleton of the boxer. The node labels are formed by each node's index and type. In turn, each node represents a skeleton branch segment, which is coloured differently from its adjacent segments and labelled according to its corresponding node index.

for computing shock graphs also suggests a way of computing the similarity between two shock graph nodes.

The idea is to use the lines fitted to the shock points which model part of the nodes' data, to measure the similarity between the nodes. We will take the line segments (the model) of one node and compute how well they approximate the data of the other node. This is analogous to asking, in a generative model approach, the conditional probability of the data coming from a given model.

Our algorithm for measuring the goodness of fit of a given model and data is similar to that with which the data is modelled. The main difference is that now we will fix the slopes of each component of the model, and will only allow the line segments to either shrink or grow to best fit the data. The assumption here is that changes in viewpoint cause parts to vary their dimensions but that their *pattern of velocities and acceleration changes* will remain roughly invariant.

Let $M(u)$ be the set of line segments that model the data points in node u. We want to compute the error of using $M(u)$ as a model for the data points in node v. To do this, we will fix the slopes and y-intercepts of the line segments in $M(u)$ and will minimize the fitting error over all combinations of inner endpoints[12]. That is, for the given ordering of line segments in $M(u)$, the first and last data points in v will determine the outer endpoints of the model, and the remaining data points will be the inner endpoints to consider. See Figure 2.8 for an example.

Let $E(M(u), v)$ be the minimum error of modelling the radius function of v with model $M(u)$. Abusing notation, we will refer to this error as $\hat{E}_M$, which is given by

$$\hat{E}_M(l_s, l_e, p_s, p_e) = \begin{cases} \text{LSE}(l_s, p_s, p_e) & \text{if } l_e - l_s = 1, \\ \min\limits_{p_s+\delta_s \le p_i \le p_e-\delta_e} \Big\{ \hat{E}_M(l_s, \lceil l_e/2 \rceil, p_s, p_i) + \hat{E}_M(\lceil l_e/2 \rceil, l_e, p_i, p_e) \Big\} & \text{otherwise;} \end{cases} \tag{2.4}$$

[12]Both model and data points are oriented such that they decrease from left to right.

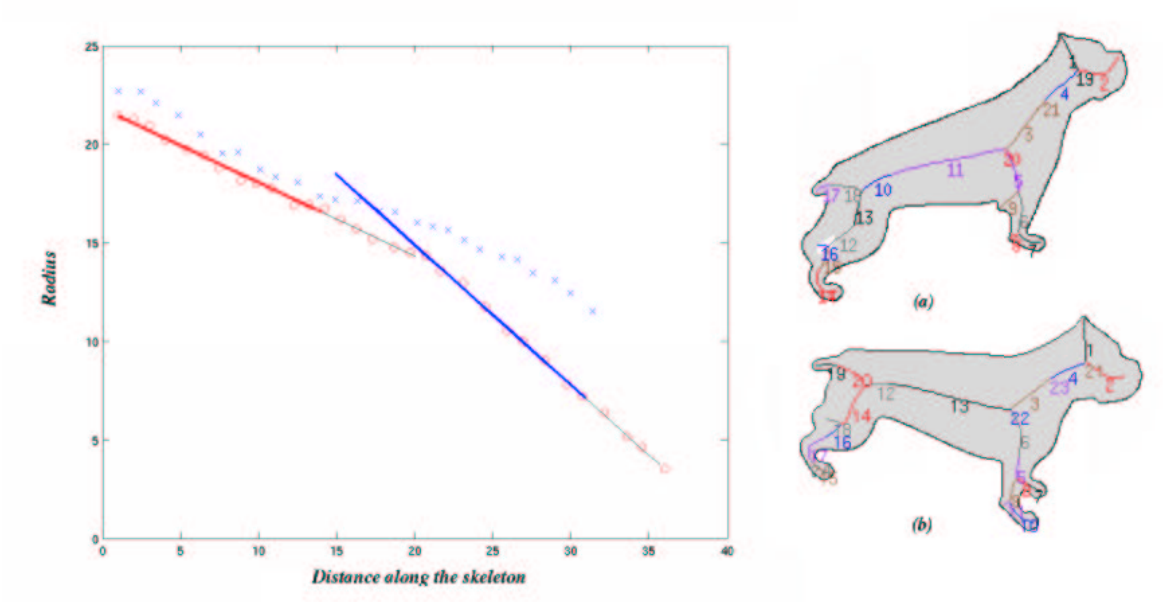


Figure 2.8: The model (2 line segments) for the skeleton segment 2 in shape **a** is used to fit the radius function of segment 2 in **b**. The data points of **a**-2 are shown in red and those of **b**-2 in blue. In this example, the two line segments have shrunk to best approximate the data. Parts of the original model are shown with a thin black line.

where l_s and $l_e - 1$ are the start and end indices of the chain of adjacent line segments in M. That is, l_e is the index of the current last segment plus one. The number of segments used to fit a sequence of points is given by $l_e - l_s$. LSE(l_s, p_s, p_e) is the least-square error of fitting points between endpoints p_s and p_e with line l_s. Finally, δ_s and δ_e require a special explanation, which is discussed next.

We want to guarantee that all the components of the model are used to fit the data and so, given that we allow each component to become shorter or longer, we must set some limits to this length variation. The obvious limit is to require that all the components be greater than zero; a better limit is to define a range for its variation. For example, when matching object parts that have been deformed due to a viewpoint change, we can expect parts to not vary their size by more than k times. This value will accommodate changes in size due to perspective, such as the legs of the boxer getting shorter as we

move from a side view to a top view. δ_s and δ_e aim to capture this idea of limits to the variation in the number of points per component. $p_s + \delta_s$ is the first point index after p_s that will ensure that there are enough data points to be fit by $\lceil l_e/2 \rceil - l_s$ line segments. Similarly, $p_e + \delta_e$ is the last index point that guarantees enough data points on the right side. Note that δ_s and δ_e will vary at each iteration of min.

Properties of the Algorithm

The combinatorial algorithms that we have presented to compute $\hat{E}$ and $\hat{E}_M$ find an optimal solution to their problems. We can afford this because of the discrete and small nature of the solution space of both problems. We can thus provide an efficient implementation by using a data structure that avoids computing their recursive functions more than once with the same parameter values. In addition, the standard input of the problems does not require a large storage size for this purpose.

It is important to note that the algorithm does not consider the absolute x-and-y coordinates of the shock points, and so is invariant to translation and rotation of the skeleton. Furthermore, by throwing away this information, we do not distinguish nodes with different curvature characteristics, which we believe is an advantage for our task of viewpoint detection and for dealing with articulating objects. Consider, for example, the bent neck of an animal drinking water and the neck of the same animal in an alert position. Our algorithm will consider the two necks to be very similar, provided that the velocities and accelerations that characterize the neck remain invariant to this sort of part articulation, and that the neck's length does not vary significantly. In our experiments, we have found that this is the case for a large number of cases of articulation and viewpoint changes.

A Measure of Similarity

The fitting error $E(M(u), v) = \hat{E}_M$ is a measure of the dissimilarity between the attributes of nodes u and v. However, we often need a measure of similarity, as is the case for our matching algorithm. This measure can be computed by some function that maps the values of $\hat{E}_M$ onto the real interval $[0, 1]$ and is inversely proportional to $\hat{E}_M$. For our experiments, we first define the non-symmetric node distance function $\hat{d} : \mathbb{N}^2 \mapsto [0, 1]$, and then derive a symmetric similarity measure from it. This distance is defined as

$$d(u, v) = \xi \min\left(\frac{E(M(u), v)}{\delta(v)\bar{R}(v)}, 1\right) + (1 - \xi)\frac{|L(u) - L(v)|}{\max(L(u), L(v))},$$

where u and v are nodes in shock graphs $\mathcal{Q}$ and $\mathcal{M}$ respectively, ξ is a convexity parameter, $\delta(w)$ is the number of shock points in node w, $L(w) = d_{\delta(w)}$ is the length of the skeleton segment, defined in terms of the distance described in Section 2.5.1, and $\bar{R}(w)$ is the average radius for all the shock points in w[13].

The first term in the above equation measures the fitting error as a function of the number of points in node v assigning a distance in the interval $[0, 1]$, which will be zero when the model of u is a perfect model for the data in v, and one when the average error is greater or equal than the average radius of node v. We use the average radius of the node simply to set a bound on the fitting error proportional to the radius function of v. The second term is inversely proportional to the lengths of the skeleton segments of each node. As it can be seen in Figure 2.6, the variation in length of corresponding skeleton segments between shapes can be large. However, when searching for similar views of an object, we can assume that the lengths would not change too much and consequently penalize nodes with large difference in length. We set $\xi = 0.85$ for our experiments, so that 85% of the nodes distance corresponds to the fitting error and only 15% represents the difference in length of the skeleton segments of u and v.

[13] If the number of shock points in either one of the nodes equals one, we compute the distance between the node attributes as $d(u, v) = \xi\frac{|\bar{R}(u) - \bar{R}(v)|}{\max(\bar{R}(u), \bar{R}(v))} + (1 - \xi)\frac{|L(u) - L(v)|}{\max(L(u), L(v))}$.

A symmetric function of similarity between the nodes is computed by $\sigma(u, v) = 1 - \text{avg}(d(u, v), d(v, u))$. Note that $\sigma(u, u) = 1$ only if the line segments are a perfect model of the data points in u.

2.7 Summary

Skeletonization algorithms have evolved to the point in which small perturbations to the shape contour do not introduce unwanted branches. In turn, skeletons can be abstracted by shock graphs, which group skeleton points into nodes that are hierarchically related according to time of formation. When the limitations of shock graphs are taken into account, they become a stable shape abstraction that can guide the search for similar shapes in a database. The additional attributes associated with every node can later be used to refine the ranking of the matches and to compute node correspondences between the query and every candidate. To this end, we believe that the behaviour of the radius function is a powerful discriminating feature that provides important flexibility in terms of viewpoint invariance and natural deformation of parts.

We have developed a shock graph computation algorithm that overcomes the sensitivity of the skeleton labelling process to noise and shape deformation. The algorithm was designed so as to represent similar views of an object captured from different viewpoints by shock graphs with similar topology and node labels. We have also proposed a node similarity function that robustly measures the distance between the attributes of two nodes, which we believe also works well under noise and part deformation. However, only indexing and matching experiments will be able to provide an exhaustive evaluation of their performance.

Chapter 3

Indexing Hierarchical Structures

3.1 Introduction

Given a query shape, the goal of indexing is to efficiently retrieve, from a large database, similar shapes that might account for the query, or some portion thereof (in the case of an occluded query or a query representing a cluttered scene). These *candidate* models will then be compared directly with the query, i.e., *verified*, to determine which candidate model best accounts for the query.

In this chapter, we will review and extend a graph-indexing algorithm proposed by Shokoufandeh and Dickinson [54, 52]. The main idea of this method is that of representing features and their interrelation as directed graphs, where the edge directions capture the hierarchical dependencies among features. The eigenvalues of the adjacency matrix of the graphs are used to capture these structural relationships in stable and low-dimensional vectors (structural indices). These vectors satisfy two goals: they form a new feature vector that can be used to prune the model database and, in turn, provide structural information that will guide the search in the verification stage (Chapter 4).

We analyzed the behaviour of the indexing algorithm on large model databases and found that a large portion of the false positives delivered to the verification stage were

due to the method for computing and accumulating the votes for the structural indices of the models. This led us to define a new vote weighting function that favours the simplest models that best explain the query. Furthermore, we developed a general vote accumulator algorithm that outperforms the one employed in [52] in terms of its discriminatory power.

From here on, we will refer to indexing techniques as those meant to prune a model database in order to provide a small set of candidates to a matching (verification) stage. Thus, the goodness of a given method will depend on: the number of candidates that need further verification, the probability of the target not being among the candidates (which should be close to zero), and the efficiency of the indexing technique relative to the matching algorithm's complexity.

3.2 Related work

Several alternatives for indexing shape features have been proposed. The most popular ones are based on decision trees, interpretation tables, hashing, nearest neighbour searching, and model prototyping.

A decision tree [42] is a mechanism for hierarchically partitioning a database. A query shape is matched to the root, and depending on the results of the match, the process is applied recursively to one of its children. At each step, the space of possible models is reduced. These methods are very sensitive to noise and occlusion, which may lead to the descent down an incorrect path.

In turn, interpretation tables [16] are related to hashing techniques, which use a hashing table and a hash function to map recovered image features to a list of models that contain those features. Each feature match is used to vote for a model. A well-known hashing technique in computer vision is geometric hashing. For instance, Lamdan et al. [37] use geometric hashing to map affine invariant *interest point* coordinates, expressed

relative to basis-triplets of three non-collinear interest points, to a set of ordered pairs of models and basis-triplets. Thus, they simultaneously collect votes for models and for the basis-triplets. This information is later used to find the affine transformation that best aligns the model to the query.

In order to account for noise in the image features, the hash function is usually required to locate *similar* features in the same bin. The main problem with hash tables is that their fixed-sized bins do not efficiently represent the non-uniform distribution of the data. This lack of adaptation gave rise to other data structures especially designed for similarity searching and for higher-dimensional feature vectors (see Section 3.3.3.)

In recent years, nearest-neighbour (NN) techniques have become very popular for feature-based retrieval applications. For instance, Beis and Lowe [4], use NN search databases to store classes of feature vectors that characterize arrangements of lines. A feature vector is formed, for example, by the angles between a set of three co-terminating lines. Other perceptual grouping methods, as well as the number of lines considered, define different feature classes. The number of relationships among lines encoded in a feature vector determines how global or local the feature is. A NN search on the corresponding index database is performed for every image feature, allowing each match to vote for a model according to the saliency of the feature in the database.

The method proposed by Shokoufandeh and Dickinson [52], which we will cover in depth in the following sections, also performs a NN search of object features as performed by Beis and Lowe. However, instead of having several index databases, one for each feature grouping, the relationships among features are chosen as the characteristic property that will help prune the database. Thus, the method provides a general technique appropriate for applications in which the relationships among features are sufficiently rich to distinguish objects in a large database.

Indexing into a database of prototypes is another technique that helps reduce the search space. The idea is to organize the database by grouping similar objects and

choosing a representative for each group that is similar enough to all the other members. This idea can be recursively applied, forming a hierarchical representation of the database. Unfortunately, it is not always possible to find clusters of similar shapes that are large-enough to significantly reduce the size of the database. This technique is well-suited for domains in which it is possible to cluster views into prototypes or to artificially create prototypes to represent all the members in a given group. Good prototypes are crucial to obtain a balanced hierarchy of small classes, which is necessary to minimize backtracking and avoid a linear search of the database.

3.3 Spectral Characterization of Graph Structure

In [54], Shokoufandeh and Dickinson consider graphs as a means of capturing relationships among features. They noticed that for certain domains, the structure of these relationships carry enough information to quickly prune the model database, and to provide strong constraints on verification. They first proposed a method for encoding the structure of undirected, unique rooted trees, which was later extended, in [52], to directed acyclic graphs (DAGs).

The topological characterization of a DAG is based on the eigenvalues of the graph's adjacency matrix, the so-called *spectrum of the graph*. Such eigenvalues are related to the branching factor, depth, and number of nodes in the graph, and so they provide a way of encoding structural properties of the graph. Indeed, the analysis of the properties of the spectra of graphs is a well-established area in graph theory (see the following books for an introduction to the subject [13, 12, 11]). In [52], the stability of the spectra of DAGs under minor perturbations in structure is studied and used to design a signature for this type of graph. In addition to the stability of the spectrum of a DAG, the spectral encoding of a graph's topology is also motivated by the invariance of the eigenvalues to permutations of subgraphs and relabellings of nodes.

3.3.1 Graph Indexing

We will now review the spectral indexing mechanism proposed in [52], and will briefly analyze its properties.

First, note that any graph can be represented as an antisymmetric $\{0, 1, -1\}$ adjacency matrix, with 1's (-1's) indicating a forward (backward) edge between adjacent nodes in the graph (and 0's on the diagonal). The goal is to map the eigenvalues of a DAG to a point in some low-dimensional space, providing a stable, compact encoding of structure.

Specifically, let G be a DAG rooted at node R_G whose maximum branching factor is $\Delta(G)$ and whose root degree is $\delta(G)$. Let the subgraphs rooted at the children of R_G be $G_1, G_2, \ldots, G_{\delta(G)}$, as shown in Figure 3.1. For each subgraph, G_i, whose root degree is $\delta(G_i)$, compute[1] the magnitudes of the eigenvalues of G_i's submatrix, sort them in decreasing order, and let S_i be the sum of the $\delta(G_i)$ largest magnitudes. The sorted S_i's become the components of a d-dimensional vector assigned to R_G, where $d = \max_k \{\Delta(G_k)\}$ is the maximum dimension of any graph in the model database. If the number of S_i's is less than $\Delta(G)$, then the vector is padded with zeroes. The procedure can be recursively repeated for the subgraph rooted at every non-terminal node, thus assigning a vector to each node in the DAG. Such a vector is called the *topological signature vector* (TSV), and is denoted by $\chi(v)$, where v is the root of a directed acyclic subgraph of G. The details of the TSV transformation, the motivation for each step, and an evaluation of its properties can be found in [52].

The above definition of the TSV has the disadvantage of ignoring some information related to terminal nodes (leaves). The TSV of a DAG, G_v rooted at node v that has terminal nodes $\{l_i\}_{i=1}^{n>0}$ as it children will be equal to that of the same graph with these leaves removed. The reason for this is that the zero eigensum of a leaf is indistinguishable

[1] We use SVD to compute the magnitudes of the eigenvalues.

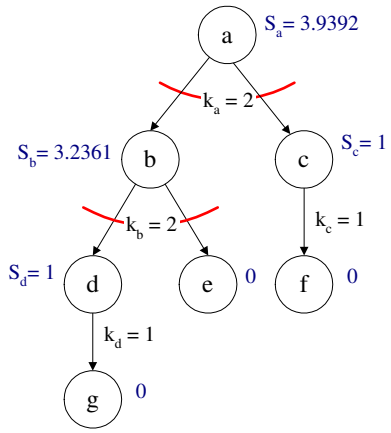


Figure 3.1: Forming the graph's signatures. Next to each node is the result of summing the $k_v = \delta(G)$ largest eigenvalues corresponding to the subgraph rooted at the node v. The TSV of node a, $\chi(a)$, is: [3.9392, 3.2361, 1], and the other indices are: $\chi(b) = [3.2361, 1, 0]$ and $\chi(c) = [1, 0, 0]$. All the indices will be padded with zeros according to the maximum branching factor in the database.

from the padded zeros of a TSV. There is a simple solution to this problem. We can add, as an extra dimension, the eigensum S_v corresponding to G_v. Since this sum will be greater than all the eigensums of the subgraphs rooted at the children of v, it will always be the first dimension of the TSV, as is the case in Figure 3.1. We followed this last procedure to compute the TSV to carry out the experiments of Chapter 5, in which the maximum dimension of any TSV in the model database was 8.

3.3.2 Dealing with PINGs

Graph spectra do not uniquely characterize graphs. In fact, it is not uncommon to find **p**airs of **i**sospectral **n**on-isomorphic **g**raphs, namely PINGs, as described in [12]. Thus, an index vector solely based on the graph's spectrum, such as the TSV, will potentially retrieve graphs that are very dissimilar to the query. This lack of uniqueness of the indices is solved in [54] by computing a TSV for every non-terminal node. Now, if we

describe a graph's structure as a collection of TSV's, the likelihood that for the subgraph rooted at each node in one graph, there exists a co-spectral graph rooted at some node in the second graph, is small. In addition, having multiple indices for different portions of the graph provides a means for dealing with missing data and/or occlusion.

The indexing algorithm is as follows: compute the indices (TSVs) for all the non-terminal model nodes in all the models and store them in a point database supporting nearest-neighbour search. Then, given a query graph, compute the indices of all the non-terminal nodes in the query and find their nearest neighbours (NN) in the index database. Finally, let the retrieved points vote for their associated model graphs and accumulate the votes as evidence, according to the following weighting function:

$$W(q,m) = \frac{||\chi(q)||}{1 + ||\chi(m) - \chi(q)||}, \tag{3.1}$$

where q and m are the query node and the model node, respectively, and $||\cdot||$ denotes the L_2-norm.

This weighting function exploits the fact that the magnitudes of the graph's eigenvalues are proportional to the branching factor and number of nodes in the graph [39]. Thus, the numerator weights the vote according to the relative size of the encoded subgraph in the query graph, which causes both larger and more complex structures to be more relevant than smaller or simpler ones. In turn, the weight is also inversely proportional to the distance or dissimilarity between the indices.

3.3.3 Indexing as a Nearest-Neighbour Search

Nearest-neighbour (NN) search in high-dimensional spaces is a well-studied subject. In the last decade, a number of techniques have evolved into a mature state. An example of the activity in this field is the work of Hjaltson and Samet [27], in which they propose a NN search algorithm based on an index data structure known as a quadtree, which organizes the space as a hierarchy of disjoint boxes. This work stimulated the development

of spatial data structures, such as R-tree, R*-tree, SS-tree, and SR-tree, among others, which more efficiently partition the space into minimum bounding hyper-**r**ectangles and/or hyper-**s**pheres. This evolution ultimately led to the recent A-tree and X-tree NN search algorithms. A complete survey of the subject can be found in [10].

The performance of the NN search techniques proposed so far is highly dependent on the characteristics of the data distribution. With the exception of approximate approaches, we can still find datasets that will result in a particular technique being outperformed by a linear search. Fortunately, it is usually possible to find an indexing method with sublinear performance for the average case for a given domain. An evaluation of several state-of-the-art NN search algorithms can be found in [19].

For the rest of the chapter, we will assume that an appropriate selection of a NN search technique has been made[2]. However, there is still one last issue to address on this matter. When performing a NN search, we can usually retrieve either the k nearest-neighbours (k-NN) of the query point or all the points within a fixed-size hypervolume centered at the query point. k-NN has the advantage of fixing the retrieved number of points regardless of their distance to the query. Unfortunately, this means that we can potentially retrieve points that are far away from the query. In addition, k is an arbitrary value that might lead to missing good points[3]. Range queries, on the other hand, eliminate the problem of missing similar points at the cost of possibly having an unbounded number of matches. For our indexing needs, considering points that are not close is wasteful, and missing similar points reduces the amount of evidence available. Therefore, from now on we will only consider range NN search queries.

[2]In our experiments, we have used the SR-tree technique proposed by Katayama and Satoh [33]. We thank Norio Katayama for the use of his SR-tree implementation.

[3]Ties among points present an additional problem that is usually solved by returning more that k points in case of ties with the k-th point.

3.4 Improvements to the candidate selection scheme

3.4.1 Weighting the evidence

As the model database gets larger, it becomes frequent to find objects composed of other objects also in the database. For example, imagine the side view of the silhouette of a person wearing a hat and riding a horse. We might have a view similar to this one in the database, and also similar *single* views of the horse, the person, and the hat. In fact, it is reasonable to think of having parts of object views in the database so as to account for errors due to under-segmentation of the image. Situations like these will not be handled well by the indexer. The reason is that the vote weighting function described above, (3.1), does not take into account how much of the query is explained by the model nor how well the model is explained by the query. Thus, it is possible that when our query view is simply a hat, the horse-man-hat model view will rank as high as a single view of a hat and vice versa.

What we need is to represent an Occam's razor-like concept into our ranking algorithm — we want the simplest model that best accounts for the query. This can be easily accomplished by redefining the vote weighting function, such that for a query point q and neighbouring model point m, we would like to increase the weight of the vote for an object model M if m represents a larger proportion of M, and similarly, we would like to increase the weight of the vote for M if q represents a larger proportion of the query. These two goals are in direct competition, and their relative merit is a function of the task domain. The idea of favouring the simplest models that best explain the query can be factored in the weighting function, which can now be specified as:

$$W(q,m) = \frac{(1-\omega)||\chi(q)||}{T_{\mathcal{Q}}(1+||\chi(m)-\chi(q)||)} + \frac{\omega||\chi(m)||}{T_{\mathcal{M}}(1+||\chi(m)-\chi(q)||)} \tag{3.2}$$

where q and m are nodes in the query graph $\mathcal{Q}$, and in the model graph $\mathcal{M}$, respectively. The weight is normalized by the sums of the TSV norms of the entire query and model graphs, i.e., $T_{\mathcal{G}(V,E)} = \sum_{v \in V} ||\chi(v)||$. The convexity parameter $\omega, 0 \leq \omega \leq 1$, is the

weighting affecting the roles of the opposing goals described above. The first term favours models that cover a larger proportion of the query, while the second favours models with more nodes accounted for. By normalizing by $T_{\mathcal{Q}}$ and $T_{\mathcal{M}}$, we obtain vote weights whose sum is in the interval $[0, 1]$ for a one-to-one vote correspondence between query nodes and model nodes. When $w = 1/2$, for example, the sum of all the one-to-one votes for a given model will be equal to one when *all* the nodes in the query and in the models are mapped, and their TSV's are equal.

In order to choose an appropriate value for ω, we can use some knowledge about the intended domain. For example, it is usually the case that the model views in the database are not occluded while, on the other hand, the query is likely to be occluded. For such cases, we can set $\omega = 3/4$ and so, 75% of the vote weight will correspond to how well the *entire* model is explained by the query, while 25% thereof would account for the relative number of matched nodes in the *possibly occluded* query.

3.4.2 Fair Counting of Votes

We are now ready to count the votes, which might seem, in principle, a trivial task: let every point in the index database vote for several model points and then select the models with the most votes. However, in order to have a fair, or truly democratic, voting process, we must ensure that no query point votes for more than one point in the same model, and that no model point receives more than one vote from the query points. In other words, given a *large number of models*, we want to find the best one-to-one assignment of votes from query points to model points for each candidate model.

Several approaches that rely on voting schemes, such as those in [54, 4], allow a many-to-many assignment of votes, supposedly under the implicit assumption that the number of false positives due to the relaxation of such a constraint will not be significant. Unfortunately, as the model database increases in size, more models that should be discarded may receive several *unfair* votes and will be sent to the verification stage,

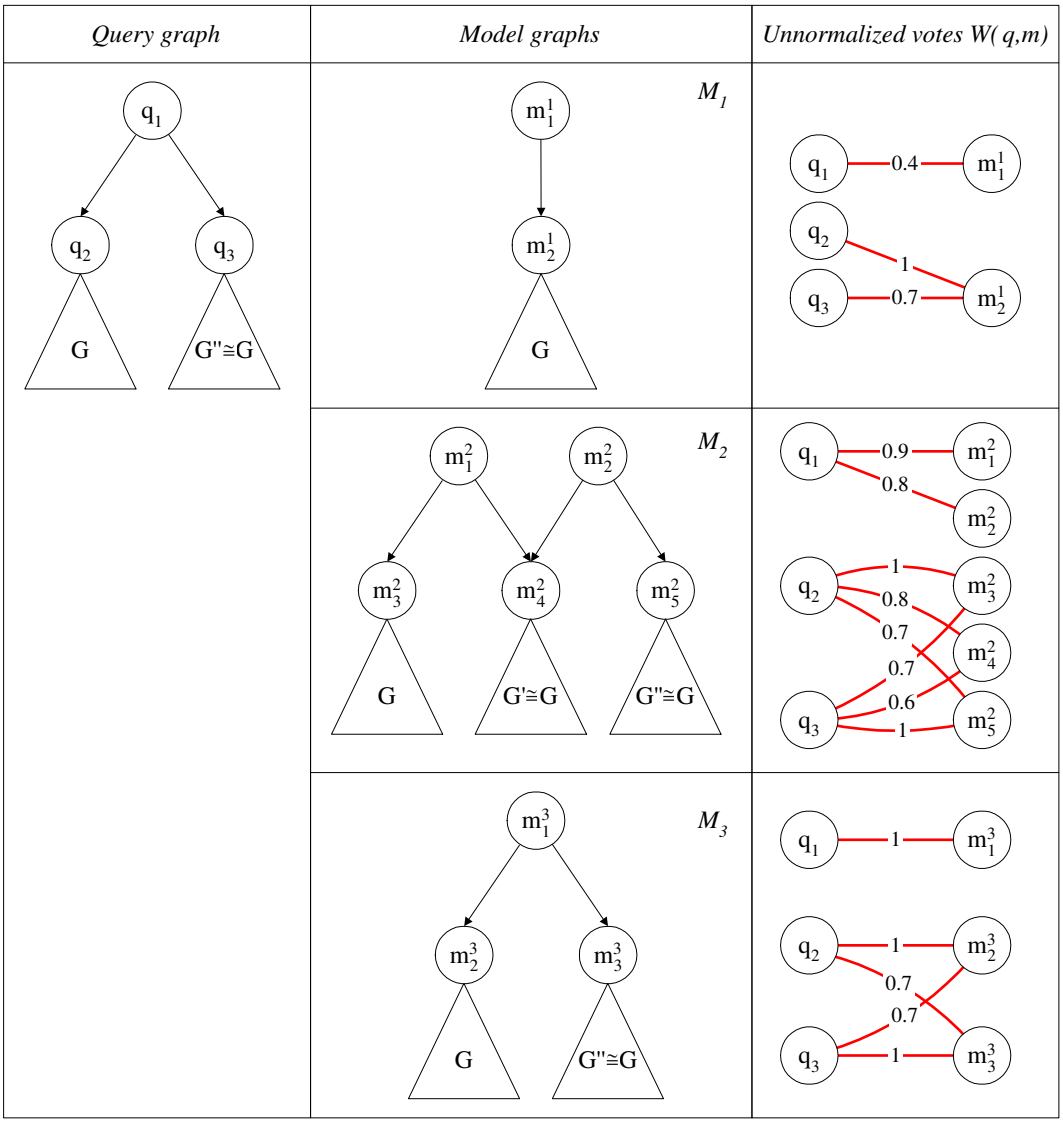


Figure 3.2: A query graph and a database with only three models. For simplicity, votes are not normalized, and consequently they do not add up to 1. Here W(q,m) represents index similarity, where perfect similarity equals 1 and total dissimilarity equals 0.

affecting the performance of the system.

3.4.3 An Example

Figure 3.2 depicts an example where a one-to-one assignment of votes would make an important difference in the selection of candidates. The task in this example is to rank the models according to the similarity of their nodes to those of the query. Here, the subgraph G rooted at q_2 is "similar" to that rooted at q_3. Imagine also that in addition to the models shown in the figure, there is a third model, M_3, isomorphic to the query graph. Note that just for this example, and in order to make the meaning of votes more intuitive, we do not normalize the votes by $T_{\mathcal{Q}}$ and $T_{\mathcal{M}_i}$ as shown in Equation 3.2 [4]. Thus, by performing a many-to-many vote accumulation, we obtain the ranking: $\{M_2 = 6.5, M_3 = 4.4, M_1 = 2.1\}$, which incorrectly orders the models. The model graph M_3 is isomorphic to the query and so we would expect it to lead the ranking. Moreover, the model graphs M_2 receives several votes due to its large size, which allows the model to collect more votes than an isomorphic graph to the query.

This particular assignment of votes is a simple case that demonstrates how misleading a trivial counting of votes can be. The reason is that a query graph may have several similar subgraphs in it, but each of these subgraphs can only *explain* at most one subgraph in a model graph, and vice versa. A one-to-one assignment of votes will turn out to be fundamental for the kind of objects we used in our experiments (see Figure 5.1). Our model database is composed of simple models represented by small graphs and by complex models that will include the graphs of the simpler models several times among their subgraphs.

[4]For simplicity and without loss of generality, we only compute the indices of the graphs' roots and their immediate children. Up to now, we have computed the indices of all the non-terminal nodes, so as to minimize the occurrence of PINGs. However, for a given domain (with minimal occlusion), it might be sufficient to consider only the indices of the nodes up to a certain level in the hierarchy, as shown in the example.

3.4.4 Small Bounded Buffers to Accumulate Votes

A simple way of obtaining a maximal one-to-one assignment of votes is to accumulate all the votes received for each candidate in a list of model graphs M_k. Each of the elements in this list is, in turn, another list of all the (q_i, m_j) pairs of votes, for $q_i \in Q$ and $m_j \in M$. We can then compute a maximum weight bipartite matching (MWBM) per model graph list, which will give us a one-to-one assignment of votes per model and ultimately the overall vote for each model.

There is one important problem with this approach. Namely, regardless of whether or not there are many-to-many assignments of votes, we will have to compute k maximum matchings, where k is the number of models that receive at least one vote, resulting in a matching complexity of order proportional in the number of votes and the number of nodes *in the models*, which is potentially large. It is clear, then, that the gain in performance of finding the optimal one-to-one assignment of votes will probably not be enough to justify the increase in time complexity of the indexing stage.

Instead, we will propose an algorithm that finds the optimal solution with a small overhead, and only for those queries that will benefit from a careful counting of votes. We will prove the optimality of the solution, and will convince ourselves that the price paid to obtain this solution is justified in terms of the gain at the verification stage.

The idea is to compute the distances between all query nodes' TSVs, and use these distances to determine which nodes might vote for two (or more) different nodes in the same model so as to prevent them to vote twice. Given the range for the NN search, we know that *query* points within this range *from each other* are potential competitors. Thus, we can give a special treatment to the sets of query nodes that are likely to produce a many-to-many mapping to model nodes. In principle, we can store all the votes corresponding to the nodes in these sets and then compute a maximum matching for each set. Since the sets are small, we are gaining something; however, we must still store all the correspondences between the nodes in the set and all the models' nodes. We

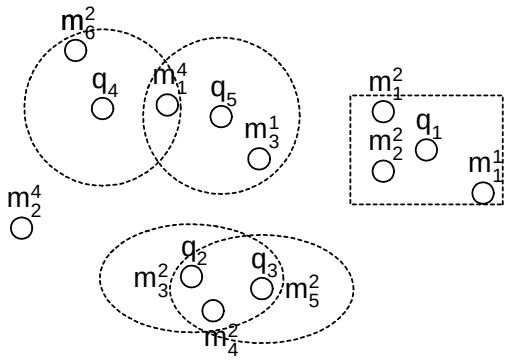


Figure 3.3: An example of overlapping index ranges for a TSV space of dimension $d = \max(\Delta(\mathcal{Q}), \max_i\{\Delta(\mathcal{M}_i)\}) = 2$. Each point v represents the 2-D TSV $\chi(v)$ of node v. In this example, the range query of nodes q_2 and q_3 is determined by a weighted L_2-norm, while that of q_1 is given by a weighted L_∞-norm. In general, a weighted L_s-norm is used to allow more variation on one dimension than on the other. The range queries for nodes q_4 and q_5, on the contrary, are specified by an unweighted L_2-norm that treats every dimension equally. The TSVs of nodes q_2 and q_3, along with those of q_4 and q_5, are said to overlap for their given range queries.

can do better by keeping only the fewest number of votes that will allow us to find the optimal solution, as we will see next.

3.4.5 Multiple One-to-One Vote Correspondence (MOOVC) Algorithm

Before we present the MOOVC algorithm, we need to define a few data structures. Let $P_1, \ldots, P_n$, be a partition of the query points' topological indices $\chi(q) \neq \vec{0}\ \forall q \in \mathcal{Q}$, such that two indices that *overlap* belong to the same partition. Given the range queries[5] $\vec{r}_1$ and $\vec{r}_2$ for two subgraphs rooted at $q_1, q_2 \in \mathcal{Q}$, the indices $\chi(q_1)$ and $\chi(q_2)$ are said

[5]A range query is usually defined by a weighted L_s distance. Given two n-dimensional vectors $\vec{x}$ and $\vec{y}$, the L_s distance between them is defined as $L_s(\vec{x}, \vec{y}) = (\sum_{i=1}^{n} w_i|\vec{x}_i - \vec{y}_i|^s)^{1/s}$. In our experiments, we use the weighted maximum distance (L_∞) for all the range queries, i.e., $L_\infty(\vec{x}, \vec{y}) = \max\{w_i|\vec{x}_i - \vec{y}_i|\}$.

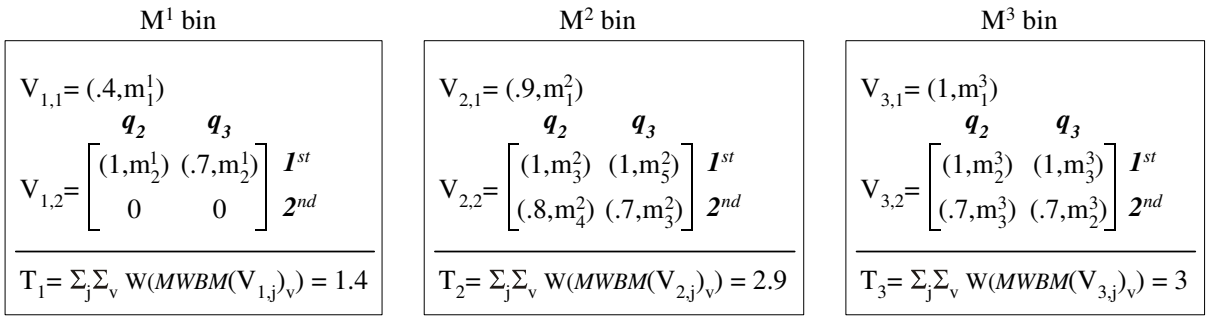


Figure 3.4: Intermediate steps of the MOOVC algorithm for the example in Figure 3.2. In the example, the range queries induce the partitions $P_1 = \{\chi(q_1)\}$ and $P_2 = \{\chi(q_2), \chi(q_3)\}$. The results, S_i, of the range queries for each query node i are $S_1 = \{\chi(m_1^1), \chi(m_1^2), \chi(m_2^2), \chi(m_1^3)\}$, $S_2 = \{\chi(m_2^1), \chi(m_3^2), \chi(m_4^2), \chi(m_5^2), \chi(m_2^3), \chi(m_3^3)\}$, and $S_3 = \{\chi(m_2^1), \chi(m_3^2), \chi(m_4^2), \chi(m_5^2), \chi(m_2^3), \chi(m_3^3)\}$.

to overlap if the d-dimensional hypervolume centred at $\chi(q_1)$ and $\chi(q_2)$ with dimensions defined by $\vec{r}_1$ and $\vec{r}_2$ intersect one another (see Figure 3.3). Let $S_{j,1}, \ldots, S_{j,k}$ be the sets of nearest-neighbour points *in the database* that are within range $\{\vec{r}_l\}_{l=1}^{k}$ of the query node indices $\{\chi(q_l)\}_{l=1}^{k}$ in the same partition P_j, for $k = |P_j|$ and $1 \leq j \leq n$. Finally, let $V_{i,j}$ be a k-by-k matrix containing in each column q the best k votes for the object model graph $\mathcal{M}_i$ received from every index $\chi(q) \in P_j$. See Figure 3.4 for an example of these matrices, which we will use to buffer the votes for each model, and for each set of overlapping indices.

We can now define the evidence accumulation algorithm. For every partition set P_j, $1 \leq j \leq n$, perform $k = |P_j|$ NN searches to find the sets of model node indices $S_1, \cdots, S_k$ corresponding to the points in P_j. Then, for every point $\chi(m) \in S_l$, where $1 \leq l \leq k$, compute the votes $W(q, m)$ for all indices $\chi(q) \in P_j$. For each vote, update the matrix $V_{i,j}$ if $W(q, m)$ is greater than the smallest vote in column q of $V_{i,j}$ [6]. Once all the nearest

[6]This sorted matrix can be implemented by k sorted lists, one for each query point. The list need not be longer that k, so whenever we are about to exceed this limit, we can safely eliminate the smallest (first) element of the list and insert the new one in the right-most position according to its value.

neighbours of the indices in P_j have been processed, compute a MWBM for each matrix $V_{i,j}$ so as to obtain the best one-to-one assignment of votes $\{(q,m)\}_{v=1}^{k' \leq |P_j|}$ from query nodes in P_j to model nodes. Finally, sum over the resulting k'-mapping of votes,

$$T_{i,j} = \sum_{v=1}^{k'} W(\text{MWBM}(V_{i,j})_v),$$

and tally a vote $T_{i,j}$ for each object model $\mathcal{M}_i$ from the partition P_j [7]. This vote is the largest sum that can be obtained by a one-to-one assignment of votes from query nodes in the partition j to the nodes in the model graph i.

After the evidence accumulation is complete, those models whose support is sufficiently high are selected as candidates for verification. The bins containing the overall votes for every model $\mathcal{M}_i$ and the vote buffer matrices $V_{i,j}$ can, in effect, be organized in a heap, requiring a maximum of $O(\log n)$ operations to maintain the heap when evidence is added, where n is the number of non-zero object accumulators.

Complexity Analysis of the Algorithm

A MWBM can be computed in $O(n^{2.5})$ [28], where n is the number of overlapping indices in a given object model. The computations of MWBMs are performed only when we encounter query nodes with similar indices. Therefore, the overhead incurred to obtain an optimal solution *for all the models*, compared to the suboptimal algorithm in [54], is $\frac{n(n-1)}{2}$ distance comparisons against $\vec{r}$ to create the partition sets, P_j, plus the computation of the small MWBMs when more than one query nodes compete to vote for the same model node. That is, the time complexity of the *overhead* is $O(m * k^{2.5})$, where m is the number of models with at least one node receiving a vote from several nodes in the same partition, and $k = max(|P_j|)$. In other words, m and k are directly proportional to the number of false positives generated by the previous method for counting votes. Hence, for little

[7] If memory requirements allow the retrieval of all the NN's of the nodes in a given partition, the votes can be efficiently computed by first sorting the retrieved points by model. This allows the matrices, $V_{i,j}$, to be processed sequentially, reducing both the storage and the heap-search requirements.

extra time spent when accumulating the evidence, we get a much larger reduction of workload in the verification stage, which is $O(n^3)$ per candidate.

Proof of the Optimality of the Solution

Proof of the optimality of the solution. By the definition of MWBM, we know that its solution is the one-to-one assignment of votes that maximizes the overall vote for the model. Then, in order to prove that the above algorithm provides an optimal solution, we just need to show that for a given model, it finds the same solution as that of computing a MWBM over the set of all the votes for the model. The competition for which model node gets a query node's vote arises only when either several model node indices $\{\chi(m)\}$ from the same model graph receive a vote from the same query index $\chi(q)$, or vice versa. By construction of the node partitions P_j, this can only happen among the indices in the same partition. Since we compute a MWBM for the retrieved weighted votes for all the indices in a given partition, we just need to show that it is enough to consider only the set of $k = |P_j|$ best votes for every node in P_j.

It is easy to prove this by contradiction. Let $T_{i,j} = \sum_{v=1}^{k} \text{MWBM}(V_{i,j})_v$ be the overall vote computed from the matrix $V_{i,j}$. Let us assume that there exists a vote $W(q, m')$, for some $q \in P_j$, that was left out of column q of $V_{i,j}$, but had it been included by using a larger buffer $V'_{i,j}$, it would have caused a rearrangement of one-to-one correspondences with a greater overall sum, $T'_{i,j} = \sum_{v=1}^{k} \text{MWBM}(V'_{i,j})_v$.

Since there is a rearrangement in the assignment of votes due to the addition of m', it must be the case that now q is mapped to m' instead of some other m^* in column q of $V_{i,j}$. Moreover, we know that among the k model nodes originally in column q of $V_{i,j}$, there is at least one model node m that will not receive a vote in the final one-to-one assignment of votes, since there are only $k-1$ query nodes other than q, which is already associated to m'. The fact that we obtain $T'_{i,j}$ without mapping m tells us that in the original vote mapping, $\text{MWBM}(V_{i,j})$, no query node q' other than, possibly q, can be

mapped to m with a vote $W(q', m) > W(q', m'')$, for $(q', m'') \in \text{MWBM}(V'_{i,j})$. Thus, since mapping q to m' results in a $T'_{i,j} > T_{i,j}$ that was not possible before, it must be the case that $W(q, m') > W(q, m)$. Therefore there is a vote greater than $W(q, m) \in V_{i,j}$ that is not in $V_{i,j}$, which contradicts the definition of $V_{i,j}$. □

It is easy to find an example to show that buffering fewer votes than the best k for every $q \in P_j$ and $k = |P_j|$ can lead to a suboptimal solution. Hence, we know that we are retaining the fewest number of votes that are needed to achieve the optimal solution to the multiple bipartite matching problems.

3.4.6 The Algorithm Applied to the Voting Example

The new vote accumulator algorithm is applied to the query and models in Figure 3.2. Let $\{\vec{r_i}\}_{i=1}^{3}$ be the vectors of range values for the NN searching algorithm that induce the partitions $P_1 = \{\chi(q_1)\}$ and $P_2 = \{\chi(q_2), \chi(q_3)\}$. Note that M_3 is a model in the database that is isomorphic to Q. By performing a many-to-many vote accumulation we obtain the ranking: $\{M_2 = 6.5, M_3 = 4.4, M_1 = 2.1\}$, which incorrectly orders the models. The results of our one-to-one accumulator algorithm are, as expected: $\{M_3 = 3, M_2 = 2.9, M_1 = 1.4\}$ [8]. The algorithm's intermediate computations are shown in Figure 3.4, where it is easy to see that the size of the vote-buffer matrices is bounded by the number of overlapping indices *in the query graph*.

3.4.7 Improving the TSV's Discriminatory Power

The discriminatory power of the indexing algorithm can be further improved by investigating new alternatives to the construction of the signature vectors. For example, up until now we have considered the edge weights in the graphs to be all equal to 1. However, we could instead use real values in the partly open interval $(0, 1]$ to express the

[8] When using normalized votes (Eq. 3.2), M_2 can be further penalized for having unmatched nodes.

strength of the link between nodes, where the weights are a function of domain-specific or domain independent properties. To this end, the relative distances of the nodes to their subgraph's root can be used, for example, to have an inverse numerical influence in the spectrum, so that noise further down in the hierarchy will contribute less to the graph's eigenvalues. This would penalize less the noise that occurs at lower levels of the hierarchy than noise higher up, where extra/missing nodes are less likely. In addition, geometrical information could also be incorporated as edge weights or as extra dimensions in the signature vectors so as to further constrain the votes. In [53], for example, the TSV is extended to incorporate contextual information specific to the domain. An extensive evaluation of such alternatives is left as future work.

3.5 Summary

The structure of a directed acyclic graph rooted at a single node can be characterized by the spectrum of the graph which, in turn, can be used to index the graph into a database. A TSV is an encoding of the spectrum that reduces the dimensionality of the graph according to the maximum out-degree in the model database. The dimensionality reduction is stable under small perturbations of the graph, but increases the ambiguity of the signature (index). Thus, the TSVs for every node are computed and used for indexing so as to account for the ambiguity of the indices while allowing partial occlusion of graph structure.

We extended the above graph indexing technique so as to penalize unmatched nodes in the models and in the query. Furthermore, we proposed a vote accumulation algorithm that efficiently finds the optimal one-to-one vote assignment, thus reducing the number of false positives delivered to the verification stage.

The discriminatory power of the indexing algorithm can be further improved by investigating new alternatives to the construction of the indices. For example, the relative

distances of the nodes to the subgraphs' roots could be used to reduce the influence of a node in the spectrum, favouring stability higher up in the hierarchy. In addition, geometrical information could be incorporated as edge weights to further constrain the votes. An extensive evaluation of the signal-to-noise ratio of such alternatives and others is left as future work.

Chapter 4

Matching Hierarchical Structures

4.1 Introduction

Hierarchical shape representations, such as coarse-to-fine decompositions, part-whole relationships, and multiscale descriptions, are frequently encoded as rooted trees or directed acyclic graphs (DAG). For our purposes, DAGs are more expressive than rooted trees because they allow nodes to have multiple parents and also make explicit the direction of the hierarchical relationships. In this chapter, we shall focus on measuring the similarity between two hierarchical structures encoded as DAGs, and on computing their node correspondences. In order to emphasize the role of these graphs in our framework, we shall refer to one of them as the query graph, $\mathcal{Q}$, and the other as the model graph, $\mathcal{M}$.

4.2 Related work

Previous work on matching hierarchical structures has assumed different constraints on the type of graphs they can handle and on the characteristics of the node correspondences. For instance, in [57], Umeyama matches same-size, weighted graphs by an eigendecomposition method that finds the permutation matrix that minimizes the difference of the graphs' arcs' weights. In [20], Gold and Rangarajan use a graduated assignment approach

to match attributed graphs without guaranteeing that all the hierarchical dependencies among nodes are preserved. In general, there are a great number of graph matching algorithms that are too restrictive to be useful in vision applications where we need to measure the similarity among attributed graphs that may not be isomorphic and may have different numbers of nodes due to noise and/or occlusion.

When considering a domain in which occlusion is non-existent and shape segmentation is exact, we still need to deal with the stability of the shape representation, i.e., perceptually similar shapes should have identical or at least very similar representations. Thus, relying on graph isomorphism for matching demands a stable representation or a very dense sampling of the object's viewing sphere to accommodate the instabilities of the representation. The complexity of the problem increases rapidly if part articulation also causes changes in the representation of the shape.

We can relax the requirements on representational stability if we allow for some degree of misleading information in the graphs such as spurious noise, clutter, occlusion and part articulation, and look for the *largest isomorphic subgraphs* that satisfy all the hierarchical constraints. To this end, Pelillo et al. [46, 47] cast the attributed tree matching problem as a maximum weight clique problem based on an auxiliary association graph by following an approach first introduced in [3, 36]. The algorithm looks for the set of nodes in the query and in the model graphs that preserve the hierarchical constraints imposed by the representation, and maximize cumulative pairwise node similarities among the nodes. The best model from a set of candidate model graphs is then selected by rank ordering the models according to the weights produced by their largest isomorphic subgraph with the query, scaled by the relative mass of this subgraph with respect to the mass of the model. One problem with this approach is that by only considering the largest isomorphic subgraph, small noise in the graph can largely affect the similarity value obtained.

In [49], Sebastian et al. address the problems of noise, articulation and deformation of parts, and occlusion for *unrooted trees* by computing the graph edit-distance. The

edit-distance is defined as the minimum cost of the deformation path that brings two shapes to a simpler common shape. Four edit operations are defined to deform a shape into another, three of which are in the graph domain and allow different types of merges and deletions of nodes. The fourth operation permits altering the nodes' attributes, thereby producing a geometrical deformation rather than a structural one. The main disadvantage of this technique is that an occluded query will dramatically slow down the algorithm for it will spend much time editing out extraneous nodes.

In order to deal with inexact graph matching, several authors have analysed the properties of the eigenvalues and eigenvectors of the adjacency matrix of a graph [51, 57, 54, 9]. The approaches proposed in [51] and [57] both have the main disadvantage of being unable to match graphs of different size. In contrast, Luo and Hancock [9] allow different-size graphs to be matched by posing the search for node correspondences as a maximum likelihood problem, in which the query graph is the observed data and the node correspondences are the hidden variables. Finally, Shokoufandeh et al. [52] propose a recursive bipartite matching algorithm, which we will analyse in detail in the following section.

4.3 DAG Matching

In [52], Shokoufandeh et al. develop an algorithm to deal with noisy DAGs under the presence of occlusion and articulation of parts. Such a method is a modified version of Reyner's algorithm [48, 58] for finding the largest common subtree. The main idea of the algorithm is to cast the structural matching problem as a set of bipartite graph matching problems. A similarity matrix for the graphs is computed, in which each entry of the matrix is the result of a *domain-dependent* node similarity, $\sigma(u, v)$, and a measure of the nodes' topological similarity. Hence, the node similarity function measures the pairwise similarity between the node's attributes and the structure of the DAG rooted at

that node. The best pairwise node correspondence obtained after a maximum cardinality maximum weight (MCMW) bipartite matching is used to split the graph in two. The split yields two pairs of smaller DAGs to match, and so the algorithm recursively proceeds thereafter in a greedy search fashion. The steps of the algorithm are sketched in Figure 4.1.

The key idea in casting a graph-matching problem as a number of MCMW bipartite matching problems is to use the topological signature vectors (TSV), computed for indexing, to penalize nodes with different underlying graph structure. Recall that the TSV of node v, denoted by $\chi(v)$, is a spectral encoding of the topology of the DAG rooted at v, and is covered in detail in Section 3.3 when investigating the indexing of hierarchical structures. The advantage of using the TSV to enforce structural similarity is that these vectors can be compared to obtain a bounded measure of similarity, which allows the matching of subgraphs with small structural differences.

We implemented Shokoufandeh et al.'s matching algorithm and evaluated it on a large database of shapes. The experiments revealed a number of problems with the approach, which we will study in the following sections, along with our proposed solutions.

4.3.1 Definitions and Notation

First, let us define the matching algorithm and its components. Let $\mathcal{Q} = (V_\mathcal{Q}, E_\mathcal{Q})$ and $\mathcal{M} = (V_\mathcal{M}, E_\mathcal{M})$ be the two DAGs to be matched, with $|V_\mathcal{Q}| = n_\mathcal{Q}$ and $|V_\mathcal{M}| = n_\mathcal{M}$. Define d to be the maximum degree of any vertex in $\mathcal{Q}$ and $\mathcal{M}$, i.e., $d = \max(\delta(\mathcal{Q}), \delta(\mathcal{M}))$. For each vertex v, let $\chi(v) \in R^d$ be the unique topological signature vector (TSV), introduced in Section 3.3. The bipartite edge weighted graph $\mathcal{G}(V_\mathcal{Q}, V_\mathcal{M}, E_\mathcal{G})$ is represented as a $n_\mathcal{Q} \times n_\mathcal{M}$ matrix $\mathbf{W}$ whose (q, m)-th entry has the value:

$$\mathbf{W}_{q,m} = \alpha\ \sigma(q, m) + (1 - \alpha)\ \left(||\chi(q) - \chi(m)||\right), \tag{4.1}$$

where $\sigma(q, m)$ denotes the domain-dependent node similarity between nodes $q \in \mathcal{Q}$ and $m \in \mathcal{M}$, and α is a convexity parameters that weights the relevance of each term. Using the scaling algorithm of Gabow and Tarjan [18], we can efficiently compute the maximum cardinality, maximum weight matching in $\mathcal{G}$ with complexity $O(|V||E|)$, resulting in a list of node correspondences between $\mathcal{Q}$ and $\mathcal{M}$, called $\mathcal{L}$, that can be ranked in decreasing order of similarity. This set of node correspondences maximizes the sum of node similarities, but does not enforce any hierarchical constraints other than the implicit ones encoded in $(||\chi(q) - \chi(m)||)$. Thus, instead of using all the node correspondences, we take a greedy approach and assume that only the first one is correct, and remove the subgraphs rooted at the selected matched pair of nodes. We now have two smaller problems of graph matching, one for the pair of removed subgraphs, and another for the two remainders of the original graphs. Both subproblems can, in turn, be solved by a recursive call of the above algorithm. The complexity of such a recursive algorithm is $O(n^3)$. Additional details and examples can be found in [52].

4.4 Problems with this approach

It turns out that splitting subgraphs at nodes with high confidence of being a good correspondence is not a strong enough constraint to guarantee that all the hierarchical relations are satisfied. Consider, for example, the graphs in Figure 4.2. After the first iteration of the matching algorithm, nodes (q_5, m_5) will be matched since their similarity is the highest one in $\mathcal{L}_1$. In the next iteration, the subgraph rooted at q_5, $\mathcal{Q}^*$, and the subgraph rooted at m_5, $\mathcal{M}^*$, as well as their corresponding complement graphs $\mathcal{Q}^c$ and $\mathcal{M}^c$, will be recursively evaluated[1]. When matching $\mathcal{Q}^c$ against $\mathcal{M}^c$, the best node correspondence according to the outlined algorithm will be (q_4, m_3). It is easy to see that this match violates the hierarchical constraints among nodes because the siblings q_4 and

[1]In the recursive call for $\mathcal{Q}^*$ and $\mathcal{M}^*$, nodes q_5 and m_5 will be in the solution set, and so they will not be evaluated again.

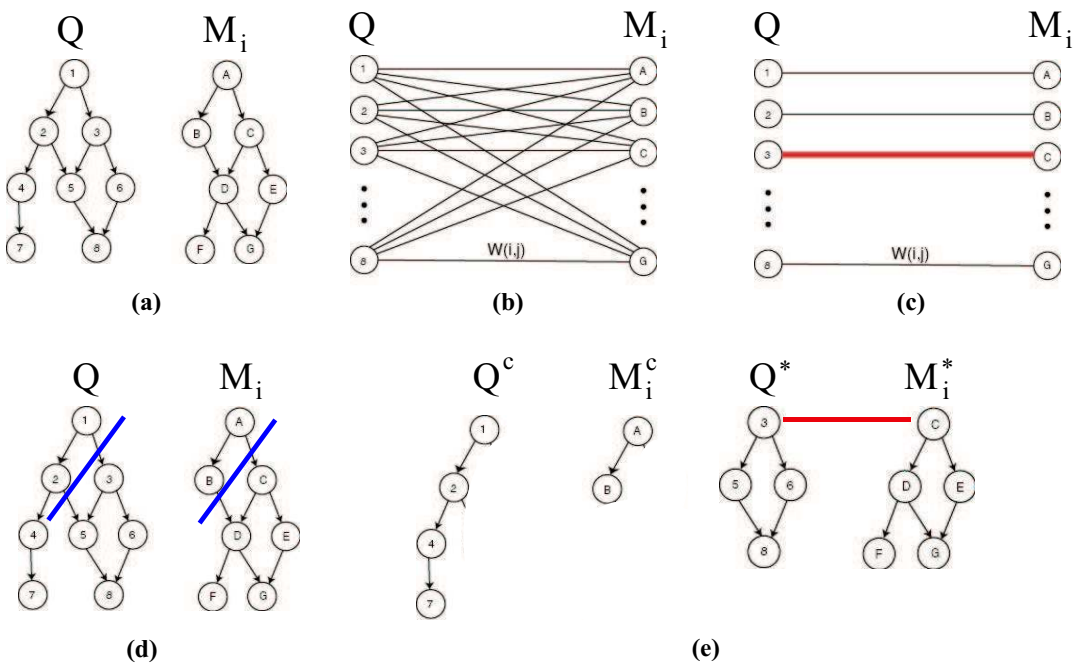


Figure 4.1: The DAG matching algorithm. (a) Given a query graph and a model graph, (b) form bipartite graph in which the edge weights are the pair-wise node similarities. Then, (c) compute a maximum matching and add the best edge to the solution set. Finally, (d) split the graphs at the matched nodes and (e) recursively descend.

q_5 are mapped to m_3 and m_5, respectively, with m_3 a parent of m_5.

Another constraint that arises in several domains is that of preserving sibling relationships. We would like that nodes with the same parent are mapped to nodes that also share the same parent (see Fig. 4.3.) Note that this constraint is not, strictly speaking, a hierarchical constraint, since there are no hierarchical dependencies among sibling nodes. It is tempting to move ahead and enforce this constraint when matching, but it turns out that there is a possibility that a sibling's relation is genuinely broken by an occluder. In such a case, we may not want to enforce this constraint or else we will be unable to find meaningful matching subgraphs. A compromise solution would be to penalize the matches that break a sibling relationship so as to favour those that provide a good set of correspondences while maintaining these relationships intact.

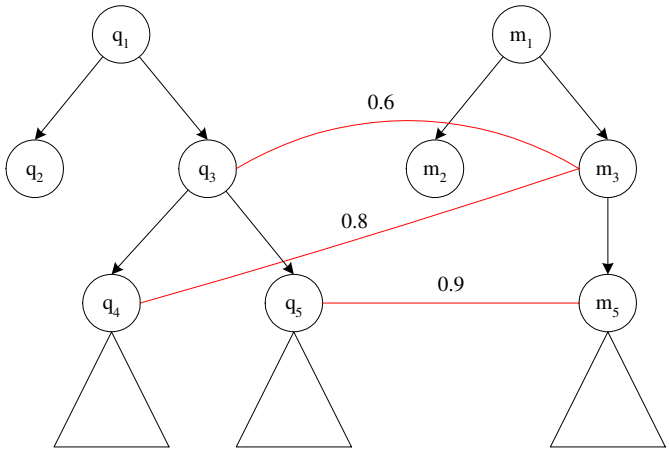


Figure 4.2: A case in which the hierarchical constraints between query nodes and model nodes will be violated after two iterations of the algorithm. Note that only non-zero $\mathbf{W}_{q,m}$ values are shown.

Figure 4.4 illustrates a situation in which the correspondence (q_4, m_5) would be the second best match. To avoid this, we can propagate the information provided by the previous best match, (q_5, m_4). This information is used to favour q_4's sibling, so that (q_4, m_3) can be chosen instead. Since we do not want to become too sensitive to noise in the graph, we shall consider preserving the sibling-or-sibling-descendent relationships instead of the stricter sibling relationship. We will refer to this asymmetric relation between nodes as the SSD relation[2]. Note that due to the asymmetry of the relation, the desired propagation of information will occur only whenever the algorithm proceeds in a top-down fashion. In the next section, we will see how to promote a top-down node matching.

Before continuing, let us define the rather intuitive node relationships that we will be working with. Let $\mathcal{G}(V, E)$ be a DAG and let u, v be two nodes in V. We say that u is a parent of v if there is an edge from u to v. Furthermore, let u be the ancestor of v if

[2]Note that while the sibling relationship is symmetric, the SSD relationship is not, i.e., if u is the "nephew" of v, then SSD(u,v) is true, but SSD(v,u) is false.

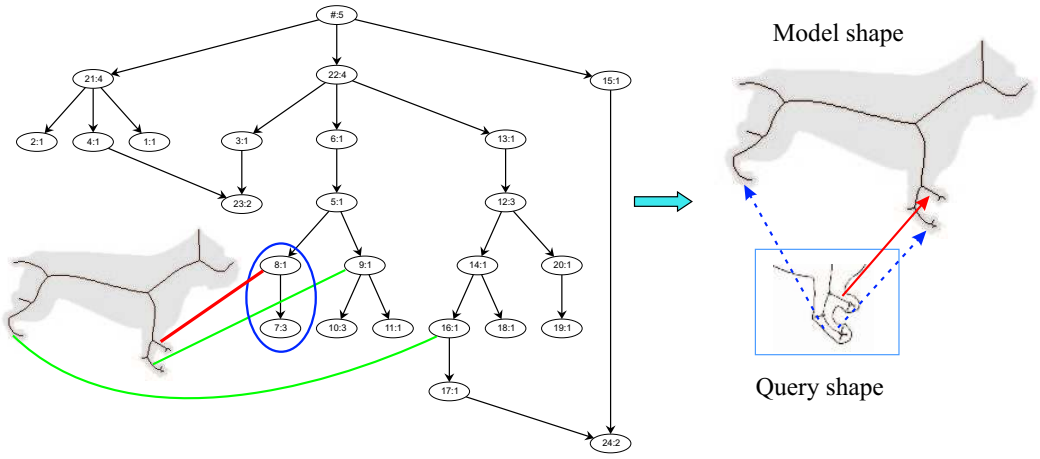


Figure 4.3: Example of the importance of the sibling relation information among nodes. Here, the left-front leg of a query view of a dog was matched to that of a similar model. When matching the right-front leg, there are two equally good candidate parts in the model: the remaining front leg and the visible back leg. In a case like this, we can use the information provided by the previous match and promote the association of parts that are locally related, i.e., share a common parent in the graph.

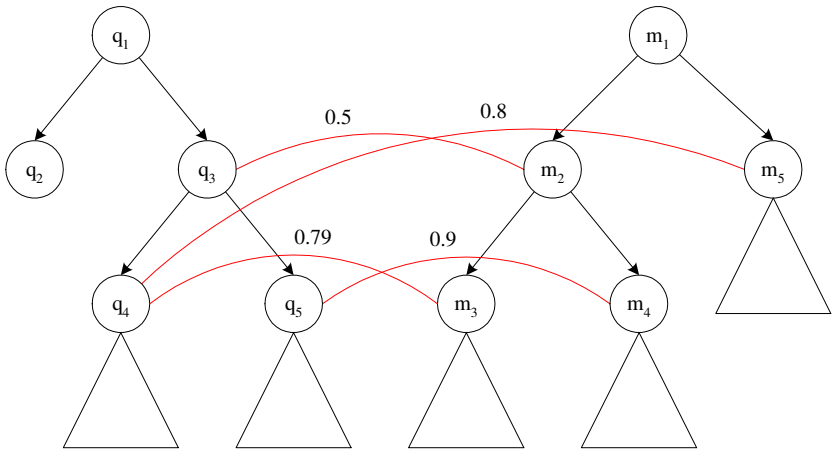


Figure 4.4: Preserving sibling relationships by propagating the information from the previous best match. The matching pair (q_4, m_3) is chosen over the slightly better match (q_4, m_5), because it results in two siblings in the query being matched to two siblings in the model.

and only if there is a path from u to v. Similarly, let u be a SSD of v if and only if there exists a parent of u that is also an ancestor of v.

The relations defined above will allow us to express the desired constraints. However, we first need to determine how to make this information explicit, for it is not immediately available from the adjacency matrices of the graphs. A simple method is to compute the transitive closure graphs of our graphs. The transitive closure of a directed graph $\mathcal{G} = (V, E)$ is a graph $\mathcal{H} = (V, F)$, with (v, w) in F if and only if there is a path from v to w in $\mathcal{G}$. The transitive closure can be computed in linear time in the size of the graph $O(|V|\,|E|)$[24].

From the above definition, it is easy to see that the transitive closure of a graph is nothing else than the ancestor relation. Computing the SSD relation, on the contrary, requires a bit of extra work. Let $\mathbf{A}_{\mathcal{G}}$ be the adjacency matrix of the DAG, $\mathcal{G}(V, E)$, and let $\mathbf{T}_{\mathcal{G}}$ be the adjacency matrix of the transitive closure graph of $\mathcal{G}$. By means of these two matrices, we can now compute the non-symmetric SSD relation by defining $\mathbf{S}_{\mathcal{G}}$ as the $|V| \times |V|$ matrix, where

$$\mathbf{S}_{\mathcal{G}}(u, v) = \begin{cases} 1 & \text{if } \exists_{w \in V}\{\mathbf{A}_{\mathcal{G}}(w, v) = 1 \wedge \mathbf{T}_{\mathcal{G}}(w, u) = 1\}, \\ 0 & \text{otherwise.} \end{cases} \tag{4.2}$$

Armed with our new matrices $\mathbf{T}_{\mathcal{Q}}, \mathbf{T}_{\mathcal{M}}, \mathbf{S}_{\mathcal{Q}}, \text{and } \mathbf{S}_{\mathcal{M}}$, we can update the similarity matrix, $\mathbf{W}$, at each iteration of the algorithm, so as to preserve the ancestor relations and to discourage breaking SSD relations. At the first iteration, $n = 0$, we start with $\mathbf{W}^0 = \mathbf{W}$. Next, let (q', m') be the best node correspondence selected at the n-th iteration of the algorithm, for $n \geq 0$. The new weights for each entry $\mathbf{W}^{n+1}_{q,m}$ of the similarity matrix, which will be used as edge weights in the bipartite graph at iteration

$n+1$, are updated according to:

$$\mathbf{W}_{q,m}^{n+1} = \begin{cases} 0 & \text{if } \mathbf{T}_{\mathcal{Q}}(q,q') \neq \mathbf{T}_{\mathcal{M}}(m,m'), \\ \beta\ \mathbf{W}_{q,m}^{n} & \text{else if } \mathbf{S}_{\mathcal{Q}}(q,q') \neq \mathbf{S}_{\mathcal{M}}(m,m'), \\ \mathbf{W}_{q,m}^{n} & \text{otherwise;} \end{cases} \tag{4.3}$$

where $0 \leq \beta \leq 1$ is a term that penalizes a pair of sibling nodes in the query being matched to a pair of non-sibling nodes in the model. It is sufficient to apply a small penalty to these cases, since the goal is simply to favour siblings over non-siblings when the similarities of the others are comparable to that of the siblings. An example of this can be seen in Figure 4.3, in which the dog's back leg and its right-front leg are equally similar to an unmatched leg in the query. The extra information we need to solve this competition is given by the sibling relationship of the dog's front legs.

It is clear that when q' and m' are the roots of the subgraphs to match, the ancestor and SSD relations will be true for all the nodes in the DAG. Thus, in practice, when matching the q'-rooted and m'-rooted DAGs, we can avoid evaluating the conditions above. In addition, we know that only a few weights will change as the result of new node correspondence, and so we only need to update those entries of the matrix. This can be done efficiently by designing a data structure that simplifies the access to the weights that are to be updated. Alternatively, the update step can also be efficiently implemented with matrices by noticing that the column of $\mathbf{A}_{\mathcal{G}}$ corresponding to node u tells us all the parents of u, while the row of $\mathbf{T}_{\mathcal{G}}$ corresponding to node v give us all the descendants of v. Thus, given a node pair (q', m') and their corresponding $\mathbf{A}_{\mathcal{Q}}$, $\mathbf{T}_{\mathcal{Q}}$, $\mathbf{A}_{\mathcal{M}}$, and $\mathbf{T}_{\mathcal{M}}$, it is straightforward to select and modify only those entries of $\mathbf{W}_{q,m}^{n}$ that need to be updated at each iteration of the algorithm.

4.4.1 Encouraging top-down matching

A careful look at the algorithm as it has been stated so far will reveal that, in general, the first node correspondences found will be those among lower-level nodes in the hierarchy. We can expect this bottom-up behaviour of the algorithm because the lower-level nodes carry less structural information and so their weight will be less affected by the structural difference of the graphs rooted at them. Therefore, nodes at the bottom of the hierarchy will tend to have high similarity values and consequently, they will be chosen to split the graphs, creating small DAG's with few constraints on the nodes.

A solution to this problem is to redefine the way we choose the best edge from the bipartite matching. Instead of simply choosing the edge with greatest weight, we will also consider the order[3] of the DAG rooted at the matched nodes to select the pair of nodes that have a large similarity weight and are also roots of large subgraphs. We define the mass, $m(v)$, of node v as the order, $n(T)$, of the DAG rooted at v. For a given graph $\mathcal{G}(V, E)$, the $|V|$-dimensional mass vector, $\mathbf{M}_\mathcal{G}$, in which each of its dimensions is the mass $m(v)$ of a distinct $v \in V$, can be computed from the transitive closure matrix, $\mathbf{T}_\mathcal{G}$, of the graph by $\mathbf{M}_\mathcal{G} = \mathbf{T}_\mathcal{G} \times \vec{1}$, where $\vec{1}$ is the $|V|$-dimensional vector whose elements are all equal to 1. Thus, $\mathbf{M}_\mathcal{G}$ is a vector in which each element $\mathbf{M}_\mathcal{G}(v)$, for $v \in V$, is the number of nodes in the DAG rooted at v.

Unfortunately, the mass does not give us enough information about the depth of the subgraph rooted at a node since, for example, the path of n nodes has the same mass as the star of n nodes. A better idea is to consider the cumulative mass, $\hat{m}$. Let $\hat{m}(v)$ be defined as the sum of all the masses of the nodes of the DAG rooted at v. Thus, the cumulative mass vector will be given by $\hat{\mathbf{M}}_\mathcal{G} = \mathbf{T}_\mathcal{G} \times \mathbf{M}_\mathcal{G}$, which can also be written as $\hat{\mathbf{M}}_\mathcal{G} = \mathbf{T}_\mathcal{G}^2 \times \vec{1}$. This vector can then be used to obtain a relative measure of how tall and wide the rooted subgraphs are with respect to the graph they belong to, by simply

[3]Here we follow the convention in the Graph Theory literature that considers the *order* of a graph to be the number of nodes in the graph, and the *size* of the graph to be the number of edges in the graph.

normalizing the masses. Let $\tilde{\mathbf{M}}_{\mathcal{G}}$ be the normalized cumulative mass vector given by

$$\tilde{\mathbf{M}}_{\mathcal{G}} = \frac{\hat{\mathbf{M}}_{\mathcal{G}}}{\underset{v \in V}{\operatorname{argmax}} \left\{ \hat{\mathbf{M}}_{\mathcal{G}}(v) \right\}}, \tag{4.4}$$

where the normalizing factor will correspond to the cumulative mass of the node whose in-degree is zero —a root— and has the greatest cumulative mass in $\mathcal{G}$.

The cumulative mass is exactly the piece of information we need, since it should be easy to see that for all the trees with n nodes, the star is the one with smallest cumulative mass, while the path is the one with the greatest. Hence, the cumulative mass, $\hat{m}$, for the root of a tree of order n satisfies $2n - 1 \leq \hat{m} \leq \frac{1}{2}n(n+1)$. This measure is a good indicator of how deep and wide a subtree is, and so provides a means to find a compromise between the node similarities and their positions in the graph.

We can then promote a top-down behaviour in the algorithm by selecting the match $(q, m)^+$ from the list, $\mathcal{L}$, returned by each MCMW bipartite matching, with the maximum convex sum of the similarity and the relative mass of the matched nodes,

$$(q, m)^+ = \underset{(q,m) \in \mathcal{L}}{\operatorname{argmax}} \left\{ \gamma \mathbf{W}_{q,m} + (1 - \gamma) \max(\tilde{\mathbf{M}}_{\mathcal{Q}}(q), \tilde{\mathbf{M}}_{\mathcal{M}}(m)) \right\}, \tag{4.5}$$

where $0 \leq \gamma \leq 1$ is a real value that controls the influence of the relative cumulative mass in selecting the best match. Since we want to promote a top-down association of nodes without distorting the actual node similarities, we suggest γ to be in the interval $[0.7, 0.9]$.

In Figure 4.5, we compare the sequences of graph splits using different values for γ. When $\gamma = 1$, we obtain the original equation in [52] that, as can be seen in the figure, tends to produce a bottom-up behaviour of the algorithm.

4.5 Selecting the best model

Given the set of node correspondences between two graphs, the final step is to compute an overall measure of graph similarity. The similarity of the query graph to the model

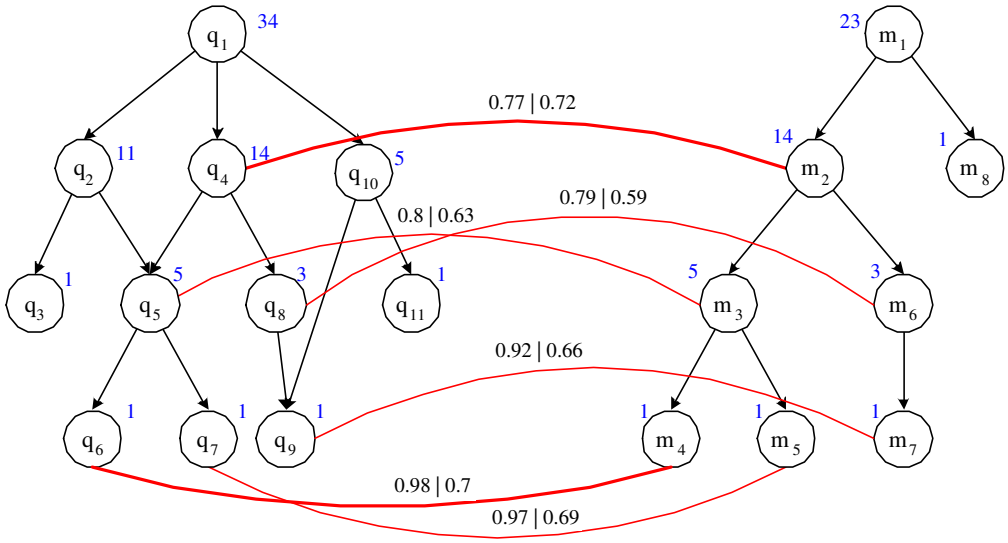


Figure 4.5: An example in which $\gamma < 1$ can promote a top-down behaviour in the algorithm. The cumulative mass of each node is shown in blue. Edge weights are computed according to Equation 4.5, for $\gamma = 1$ and $\gamma = 0.7$. For the given set of node similarities and $\gamma = 1$, the best node correspondence at this iteration is the pair of leaves (q_6, m_4), whereas for $\gamma = 0.7$, the best node correspondence is the pair of non-terminal nodes (q_4, m_2).

graph is given by

$$\sigma_\Phi(\mathcal{Q}, \mathcal{M}) = \frac{(n_\mathcal{Q} + n_\mathcal{M}) \sum_{(q,m)^+ \in \Phi} \mathbf{W}_{q,m}}{2 n_\mathcal{Q} n_\mathcal{M}}, \tag{4.6}$$

where $n_\mathcal{Q}$ and $n_\mathcal{M}$ are the orders of the query graph and the model graph respectively.

The graph similarity is given by a weighted average of the number of matched nodes in the query and in the model, where the weights are given by the node similarity of each matched node pair. If all the query nodes are matched with similarity 1, i.e., their attributes are identical, we have $\sum_{(q,m)^+ \in \Phi} \mathbf{W}_{q,m} = n_\mathcal{Q}$, and so $\sigma_\Phi(\mathcal{Q}, \mathcal{M}) = \frac{1}{2}(\frac{n_\mathcal{Q}}{n_\mathcal{M}} + 1)$. Since all query nodes have been matched, we know that $n_\mathcal{M} \geq n_\mathcal{Q}$, and so $\sigma_\Phi(\mathcal{Q}, \mathcal{M})$ will be one when all the model nodes are mapped, and less than one otherwise. Therefore, the graph similarity is proportional to the quality of each pair of node correspondences,

```
procedure isomorphism(Q,M)
    Φ(Q, M) ← ∅ ;solution set
    compute the n_Q × n_M weight matrix W^0 from Eq. 4.1
    T_Q = compute transitive closure matrix from A_Q
    T_M = compute transitive closure matrix from A_M
    S_Q = compute SSD matrix from A_Q and T_Q
    S_M = compute SSD matrix from A_M and T_M
    ;the TSV at each node were already computed at indexing time
    unmark all nodes in Q and in M
    call match(root(Q),root(M))
    return(σ_Φ(Q, M)) (see Section 4.5)
end

procedure match(u,v)
    do
        {
        let Q_u ← u rooted unmarked subgraph of Q
        let M_v ← v rooted unmarked subgraph of M
        L ← max cardinality, max weight bipartite matching between
                unmarked nodes in G(V_{Q_u}, V_{M_v}) with weights from W^{n+1} (see [18])
        (u', v') ← choose max weight pair in L from Eq. 4.5
        Φ(Q, M) ← Φ(Q, M) ∪ {(u', v')}
        update the similarity matrix W^{n+1} according to Eq. 4.3
        mark u'
        mark v'
        call match(u',v')
        call match(u,v)
        }
    while (Q_u ≠ ∅ and M_v ≠ ∅)
```

Figure 4.6: Algorithm for Matching Two Hierarchical Structures

and inversely proportional to the number of unmatched nodes, both in the query and in the model. Hence, a model that contains the query as a relatively small subgraph is not as good a match as a model for which most of nodes match those of the query graph, and vice versa.

4.6 Complexity Analysis

The final algorithm is shown in Figure 4.6. The first step of the algorithm is to compute a node similarity matrix, the transitive closure matrices, and the sibling matrices for both graphs. Assuming a linear algorithm for the pairwise node similarities, the former matrix can be computed in $O(n^3)$. The other matrices can, in turn, be obtained in linear time and in quadratic time, respectively. At each iteration of the algorithm, we have to compute a MCMW bipartite matching, sort its output, and update the similarity

matrix. The complexity at each step is then determined by that of the bipartite matching algorithm, $O(|V||E|)$, since it is the most complex operation of the three. The number of iterations is bounded by $min(n_{\mathcal{Q}}, n_{\mathcal{M}})$, and so the overall complexity of the algorithm is $O(n^3)$. Hence, we have provided the algorithm with important properties for the matching process while maintaining its original complexity.

4.7 Summary

We have extended the DAG matching algorithm introduced in [52] by eliminating the possibility of hierarchical inversions in the node correspondences and by propagating information to preserve the relationship among sibling nodes. In addition, we extended the algorithm by encouraging a top-down selection of correspondences, which increases the number of hierarchical constraints available at each iteration of the algorithm. The combination of all these extensions, coupled with our measure of graph similarity, will allow us to accurately rank the candidate graphs delivered by the indexing mechanism. It should be noted that the presented algorithm can handle graphs of different sizes that are only partly similar to that of the query.

Chapter 5

Experiments and Results

5.1 Experiments Set-up

We have systematically tested our integrated framework using both occluded and unoccluded queries. With a large number of trials on a database of 2688 graphs, this represents one of the most comprehensive set of shock graph experiments to date. Our database consists of views computed from 3-D graphics models obtained from the public domain. Each model is centered in a uniformly tessellated view sphere[1], and a silhouette is generated for each cell in the tessellation. A shape skeleton is computed for each silhouette with the algorithm proposed in [15], and its shock graph is generated as discussed in Section 2.5. Then, each graph is added to the model database, while the signature vectors of its nodes are added to the index database, as described in Section 3.3. The 3-D models that give rise to the views in the database are shown in Figure 5.1.

In the first set of experiments, we evaluate the performance of the system on a set of unoccluded queries to an object view database. The database contains 2688 views describing 21 objects (128 uniformly sampled views per object). We then remove each view from the database and use it as a query to the remaining views. For each node of

[1]We compute a perspective projection of the 3-D object onto the image plane corresponding to each viewpoint on the sphere.

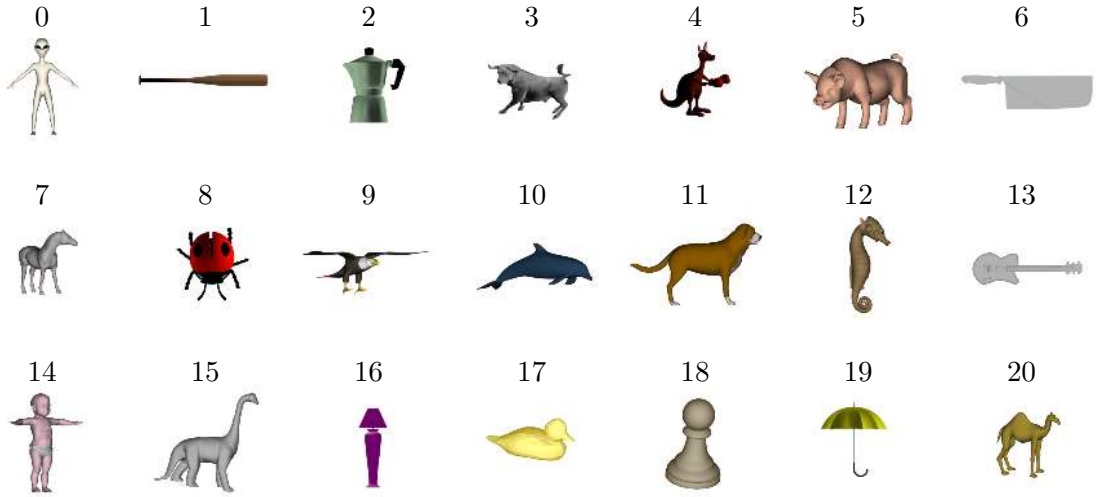


Figure 5.1: The twenty-one 3-D models use to generate a database of 2688 views. The objects are chosen so as to present a large variation in terms of shape complexity. Simple objects will be represented by small graphs that will also be subgraphs of larger graphs in the database.

the query DAG, the indexing module (see Section 3.3) will return all the TSV's within a radius of 40% of each dimension of the query node's TSV. This parameter determines the range, $\vec{r}$, discussed in Section 3.3.3, and is set so as to allow small structural differences between the query and the models.

Evidence for models is then accumulated by an efficient one-to-one mapping, as described in Section 3.4.2, and weighted according to Equation 3.2, with $\omega = 0.5$. The indexer will return at most the highest scoring K candidates, which will be next matched to the query and sorted according to similarity using the matching algorithm described in Section 4.3. The parameters for the matcher are set as follows. The node similarity weight mixing term: $\alpha = 0.7$, the penalty term for breaking SSD relations: $\beta = 0.8$, and the influence term of the nodes' cumulative masses that promotes a top-down node matching: $\gamma = 0.7$. We verified that the overall results were not sensitive to the precise choice of these parameters.

5.1.1 Evaluation Scheme

Our goal is to evaluate the framework for the tasks of object recognition and pose estimation. In addition, we want to find empirically, K, the number of candidates returned by the indexer to be considered, so as to achieve a good compromise between recognition performance and matching efficiency. The success of the object recognition task is measured as follows. If the query object (from which the query view was drawn) is the same as the model object (from which the most similar candidate view is drawn), recognition is said to be successful, i.e., the object label is correct for the first ranked view after matching K candidate views[2].

Evaluating the success of the pose estimation task is more complicated. Although we sample the entire viewing sphere, views on one side of the viewing sphere are similar to views on the other side. This similarity is more pronounced as the viewing distance grows with respect to the object's depth, i.e., the camera geometry approaches orthographic projection. Moreover, additional object planes of reflective symmetry or axes of rotational symmetry introduce additional ambiguous views with respect to pose. Therefore, the obvious method of evaluating pose estimation success by checking that the top-ranked candidate is one of the query's nine neighbours (see Figure 5.2) is somewhat harsh. Even without symmetry planes or axes, the nine neighbours of the query's counterpart on the other side of the sphere (which is *not* removed), will be similar, with the query's counterpart possibly being more similar to the query than the query's own immediate neighbours.

Properly evaluating the pose estimation requires knowing exactly which views are ambiguous due to symmetry, a costly ground-truth exercise. We therefore define a slightly weaker measure of success and require only that at least one of the 9 closest neighbours of the query on the viewing sphere is ranked before any other object's view. The assumption

[2]Note that if multiple views (perhaps from different objects) are tied for first place (i.e., "most similar"), then each can be considered to be "most similar."

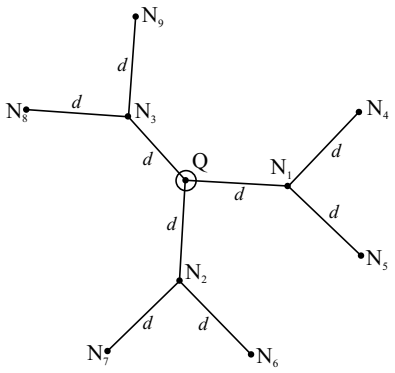


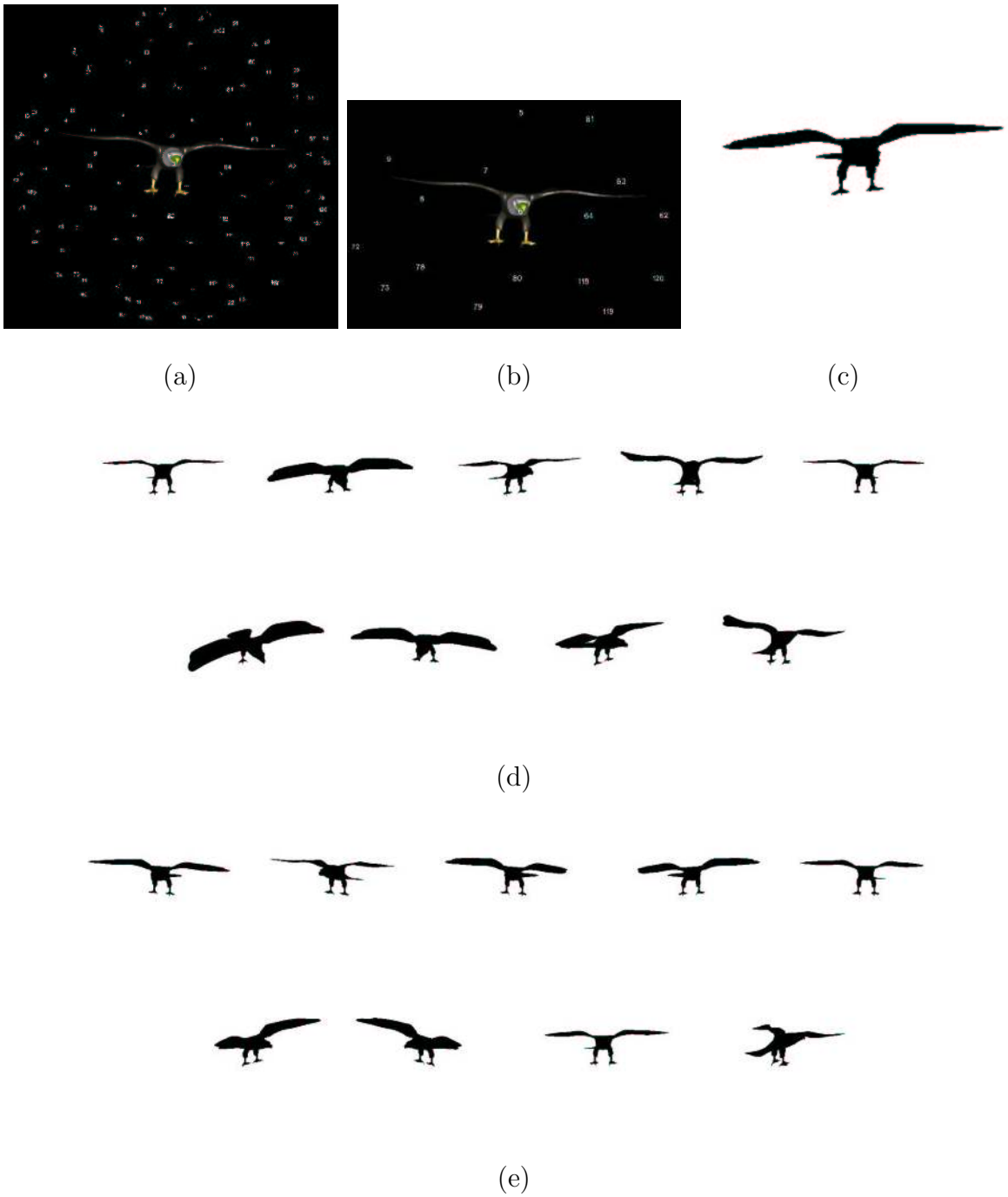
Figure 5.2: Configuration of the 9 closest neighbours of Q on the view sphere.

here is that any of the query view's non-neighbouring views (of the same object) that rank higher than the closest of the query view's nine neighbours are similar and are due to object or viewing symmetry. We will see later on that when we plot the rank of the closest neighbour among the ranked candidates, the closest neighbour scores consistently high, reflecting high pose estimation accuracy.

An alternative to the above scheme to deal with the effect of symmetries would be to remove from the database the redundant views of each object related to symmetry. However, this would require a manual intervention to decide what views to discard for each particular object. In contrast, by making no assumptions about the object views, we can test that the framework behaves as expected, and then decide on the most convenient view clustering technique that will not only provide a good solution to the issue of symmetry, but will also largely reduce the degree of ambiguity in the database which, in turn, will improve the indexing results.

5.1.2 Node correspondences

An automatic way of evaluating node correspondences has yet to be devised, and so we have not evaluated this task exhaustively. As a brief analysis and demonstration, we present in Figure 5.4 a set of cases in which the system has succeeded and others in which



(a) (b) (c)

(d)

(e)

Figure 5.3: (a) viewing sphere of the eagle. (b) close-up of the sphere. (c) view #7 of the eagle, used as query. (d) the 9 closest neighbours of view #7 in the viewing sphere, views 6, 5, 8, 80, 64, 10, 61, 9, 78. (e) The top 9 best matches returned by the matcher, views 63, 62, 111, 87, 112, 89, 105, 86, 24.

it has failed. These examples show the ability of the framework to deal with rotated, deformed, and self-occluded shapes. Note, for instance, that all the figures among the examples have parts that have disappeared, shrunk, or enlarged. The last two cases at the bottom of the page are failures. The first one shows the top-front view of the dinosaur with self-occluded neck and head being matched to a side-view of the dog. In the second case, the lack of constraints in terms of skeleton curvature similarity and relative orientation of parts allows the top view of the dinosaur to match a top-left view of the seahorse.

Some of the correct object identifications in Figure 5.4 also have a few incorrect mappings of parts. For example, the front leg of the dog is matched to its back leg due to the large deformation suffered by all its legs. In the case of the guitar, we can see that a missing branch in the skeleton led to a twist in the correspondences. It is particularly interesting to note the case of the bull, in which one of the front legs is matched to the bull's tail. It is clear that the largely deformed tail is very similar to the leg. Probably the only solution to these types of incorrect mappings is to factor in both ordering and relative orientation of the parts in the matching process and in the shape representation. In general, we have found that most of the node correspondences are assigned correctly, which allows, for example, a post-matching verification stage to exploit part order to eliminate views that are rotated or mirrored with respect to the query.

5.1.3 Limitations

Shock graphs, in principle, are scale invariant. However, the sensitivity to variation in the scale of a shape has not been thoroughly studied in the literature to date. Furthermore, significant scale differences may cause skeleton branches to emerge or disappear. To avoid introducing these kinds of problems in the experiments, we have fixed the scale of the images so that all the objects' views are generated such that they fit an image size of 300 by 300 pixels.

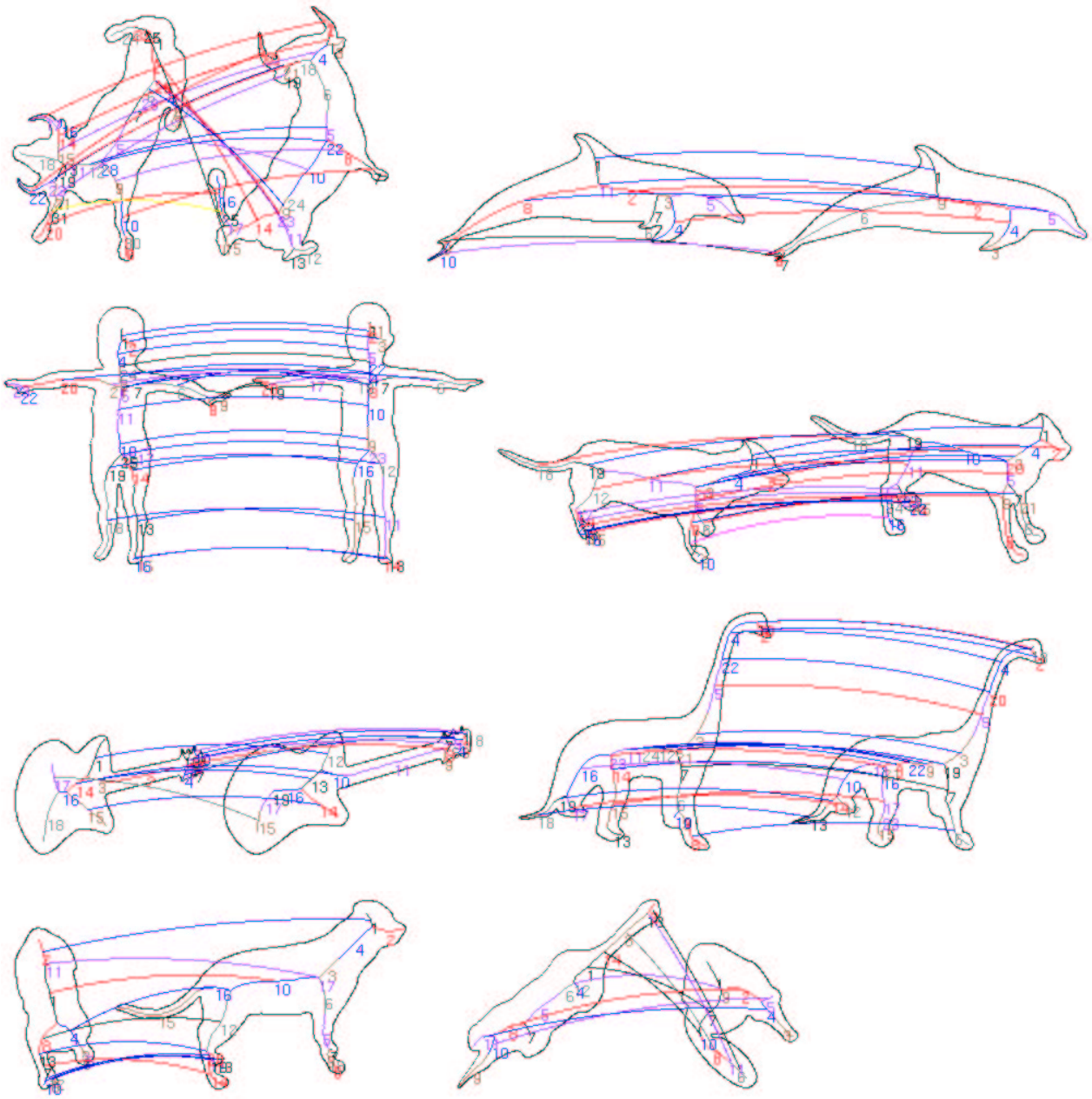
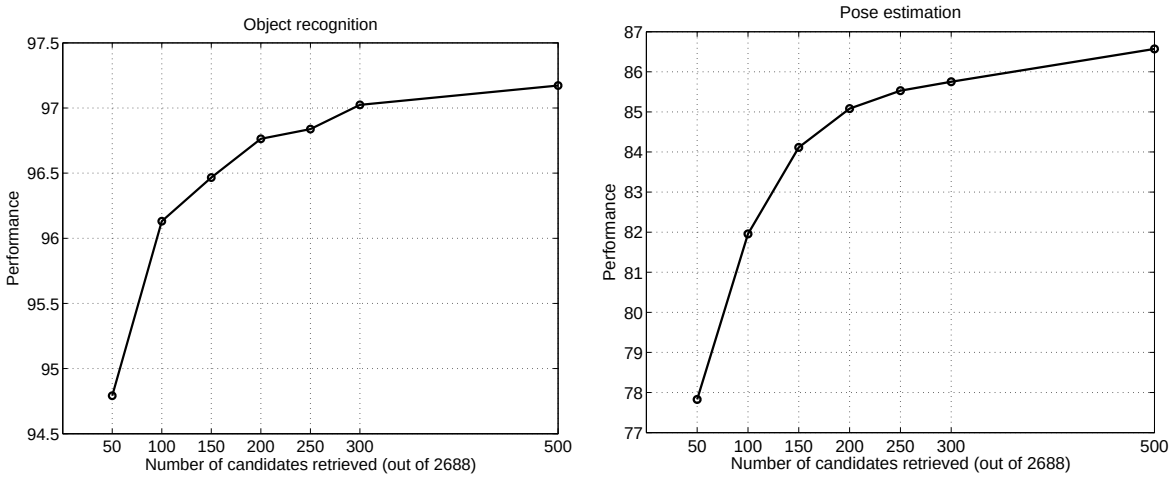


Figure 5.4: Some examples of node correspondences found by the matcher. The line colours encode the node similarity computed according to Equation 4.1, where: green = 1, blue ≥ .9, red ≥ .8, violet ≥ .7, black ≥ .6, dark grey ≥ .5, medium grey ≥ .4, light grey ≥ .3, pink ≥ .2, yellow ≥ .1



(a) (b)

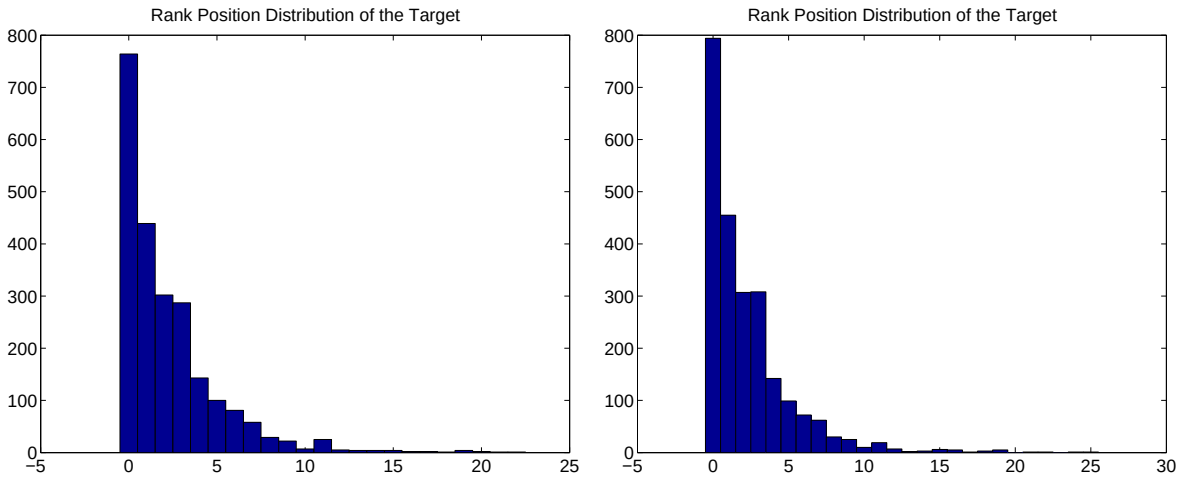
Figure 5.5: Recognition Performance: (a) Object recognition performance as a function of K; (b) Pose estimation performance as a function of K. When $K = 2688$, i.e., no indexing mechanism, the object recognition performance is 97.1726 and the pose estimation performance is 87.7604.

5.2 Object Recognition and Pose Estimation

Figure 5.5 (a) plots recognition performance as a function of increasing number of candidates returned by the indexer, while (b) plots this same comparison for the task of pose estimation. Recognition performance is very high, with better than 95% success even when we only analyze 50 candidates, thus saving 2638 matching operations. The performance on the more difficult task of pose estimation[3] is also very good. For this case, we can see that considering only 200 candidates results in a good compromise between accuracy and efficiency. The accuracy of the pose estimation is high, especially if we consider that, in general, every object has some number of degenerate views[4], and many objects have reflectional or rotational symmetry. The remaining experiments in

[3]Pose estimation often involves matching views with a large degree of deformation and self-occlusion of parts.

[4]A degenerate view is one that has no similar neighbours on the viewing sphere.



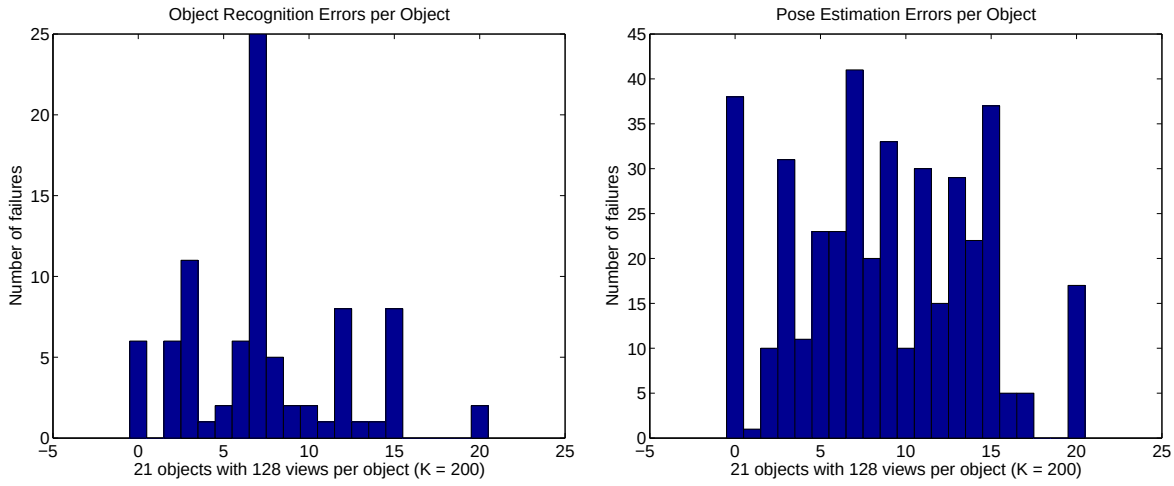
(a) K = 200 (b) K = 2688

Figure 5.6: Distribution of ranking positions for the best-ranked neighbouring view of each of the 2688 query views. All the views that rank better than the best-ranked neighbouring view belong to the same object as that of the query.

this chapter were carried out with $K = 200$.

Figures 5.6 (a) and (b) plot the distribution of ranking positions for the successfully matched, best-ranked neighbouring view of each query, for $K = 200$ and $K = 2688$, respectively. As we described in Section 5.1.1, a neighbouring view is considered to match the query view successfully if it ranks before any view of an incorrect object. Since shock graphs are meant to be invariant to permutations of their subgraphs, views at opposing sides of a plane of symmetry of the object may be more similar than the immediate neighbours of each view and the neighbours of a symmetric view might be as similar to the query view as the query's true neighbours[5]. The results confirm that only a few neighbouring views are not at the top of the ranking, which would correspond to objects that have rotational symmetry (the baseball bat and the lamp) or other types of symmetries.

[5]For example, the view at the opposing side on the viewing sphere to that of the query, plus its 9 neighbouring views, could potentially lead the ranking.



(a) (b)

Figure 5.7: Recognition Performance: (a) Object recognition errors per object; (b) Pose estimation errors per object. Each column corresponds to an object and is ordered by the rows in Figure 5.1

Errors per Object

In an effort to understand the performance, we plot in Figures 5.7 (a) and (b) the distribution of recognition and pose estimation errors across objects. It is clear from the plots that the errors are not uniformly distributed and so, in order to focus our efforts to improve performance, we need to study the cases that accumulate most of the errors.

Upon examining the results, we found, for example, that most of the object recognition errors (for $K = 200$) for the horse (object 7) and the bull (3), which both lack a plane of symmetry, were due to their query views being incorrectly matched to views of different four legged animals, such as the camel (20), the dog (11), and the dinosaur (15). In terms of pose estimation errors, objects such as the kangaroo (4), which suffered from self-occlusion of parts under small variations of viewpoint, were highly affected. Figure 5.8 shows some examples.

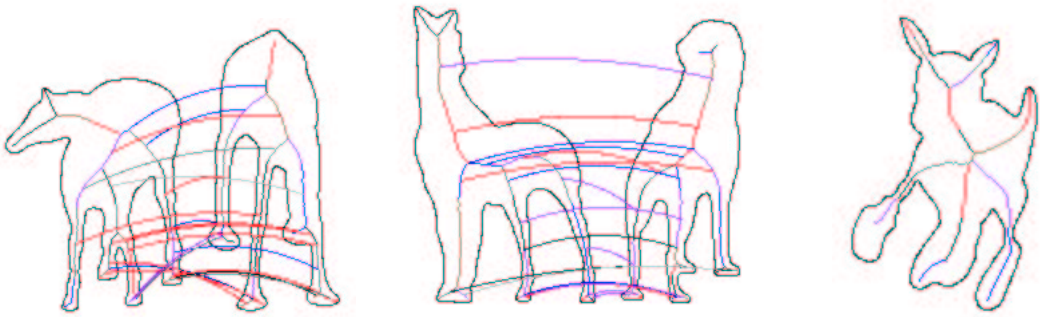


Figure 5.8: The first two images show views of the horse that failed the object recognition test. The last figure is a view of the kangaroo that failed in the pose estimation experiment.

5.3 Comparison Against Previous Approach

The results we have obtained so far allow us to evaluate the extended indexing and matching framework with respect to the original framework proposed in [54]. Figure 5.9 shows a comparison table for each of the most relevant extensions to the framework proposed in this thesis. In column **a**, the indexing votes are weighted according to Eq. 3.1 instead of Eq. 3.2, while requiring a one-to-one correspondence for the nodes in each model graph. Column **b** shows the performance when a one-to-one vote correspondence is not enforced and the votes are weighted according to the improved function, Eq. 3.2. Column **c** shows the performance of the indexing algorithm without either component, i.e., a many-to-many assignment of votes is allowed, and each vote is weighted by Eq. 3.1, as was originally proposed in [54]. These results reveal that a one-to-one vote correspondence produces much better indexing performance, and that Eq. 3.2 is particularly important if a one-to-one vote correspondence is not ensured.

In Section 4.4, a number of extensions to the matching algorithm were proposed with the main goal of eliminating violations of the hierarchical constraints from the node correspondences delivered by the matcher. By adding such extensions, we expect to eliminate the inversions while maintaining or, in the best case, improving the matching

	Indexer			Matcher			
$K = 200$	**a**. Vote weight	**b**. 1-to-1 votes	**c**. Both **a** and **b**	**d**. Anc. Rel.	**e**. **d** and SSD R.	**f**. Top-down	**g**. All **d** to **f**
Obj. Rec.	96.80	92.63	73.47	96.28	96.24	96.76	96.73
Pose Est.	84.11	72.84	53.53	84.19	83.85	85.71	84.90

Figure 5.9: Comparison table for each new feature in the indexer and in the matcher. Each column shows the performance of the extended framework without one or more of the proposed extensions, so as to isolate the individual contribution of each feature. The performance of the algorithm when all the new features are used for $K = 200$ is 96.76 for object recognition and 85.08 for pose estimation.

performance. Columns **d**-**g** evaluate the performance of the matcher when these extensions are not used. The numbers in these columns tell us that the freedom to violate the ancestor relation (col. **d**), the lack of encouragement to preserve the SSD relations (col. **e**) coupled with the freedom to violate ancestor relations, and the promotion of a top-down matching behaviour (col. **f**), do not significantly affect the matching performance. Column **g** shows that when none of the three extensions to eliminate inversions are used, the qualitative ranking of candidates remains the same. A comparison of the improved accuracy of node correspondences due to these extensions is left as future work.

The algorithm for computing shock graphs as well as the node similarity function presented in Section 2.5 also deserve their own comparison against previous approaches. Unfortunately, the algorithms used to compute shock graphs in the related work are not well documented and so we are unable to make a significant comparison. We leave as future work a comprehensive comparison of the node similarity functions in the literature.

5.4 Performance under varying sampling resolution

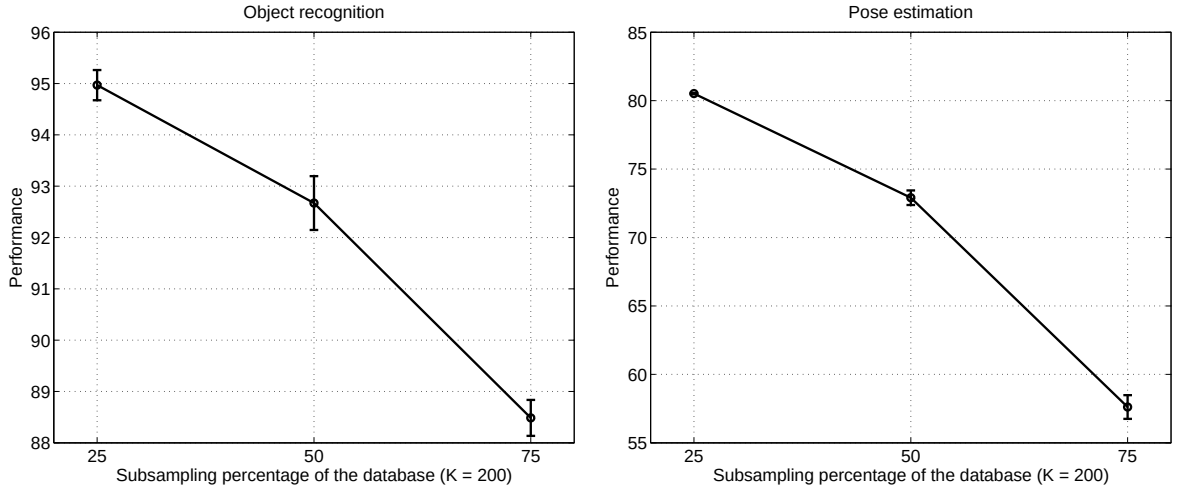
We examine performance as a function of sampling resolution of the viewing sphere by subsampling the database so as to eliminate 25%, 50%, and 75% of the views. This experiment is meant to estimate a worst-case performance for the tasks of object recognition and pose estimation having fewer views per object in the model database. The subsampling is done using a simple strategy. For each object, we generated subsampled view sets of size 96, 64, and 32 by randomly eliminating views on the object's viewing sphere. Then, we use every view in the original database with 128 views per object as a query[6] against each subsampled database. We conducted four trials of the experiment and computed the average performance. It should be noted that a random strategy for subsampling is clearly inefficient since it does not focus on eliminating redundant views from the database. A clustering technique, on the other hand, could use the proposed shock graph similarity function to perform a more clever subsampling of an object's viewing sphere.

Figures 5.10 (a) and (b) plot recognition performance as a function of sampling resolution. We can see that the performance on recognition and pose estimation is still good when sampling resolution drops to 64 views per object (over the entire view sphere). Furthermore, when the resolution drops to 32 views per object, the decrease of performance is more pronounced but still degrades gradually, showing the stability of the approach. The experiment demonstrates both the efficacy of the recognition framework and the viewpoint invariance of the shock graph, respectively.

5.5 Performance under varying numbers of objects

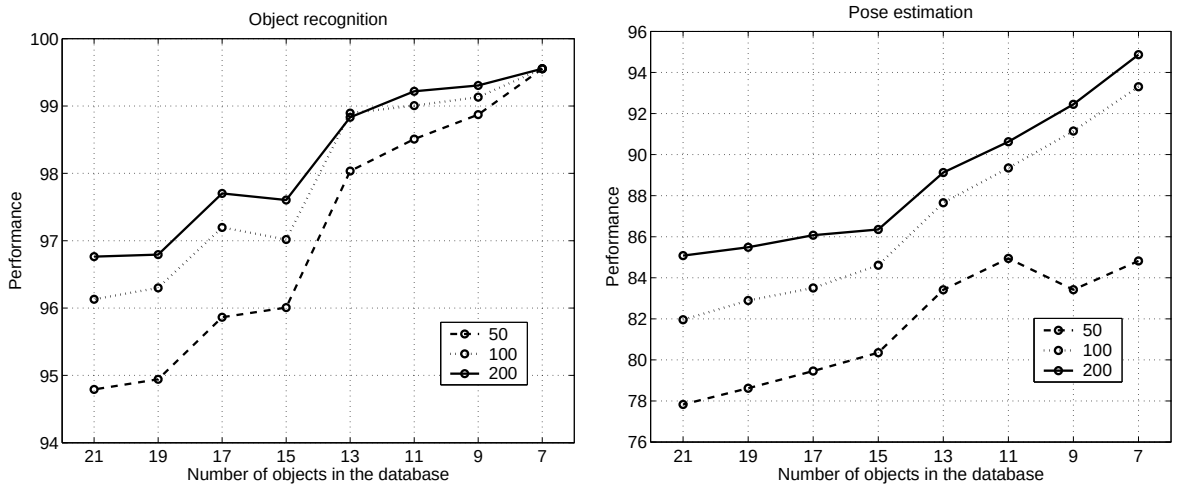
Starting with our database of 21 objects, we gradually reduced the number of objects in the database by removing pairs of objects. We started with the alien and the baseball

[6]The query view is also removed from the subsampled model database if it is still present.



(a) (b)

Figure 5.10: Recognition Performance: (a) Object recognition performance as a function of sampling resolution; (b) Pose estimation performance as a function of sampling resolution.



(a) (b)

Figure 5.11: Recognition Performance: (a) Object recognition performance as a function of number of objects and K; (b) Pose estimation performance as a function of number of objects and K. The non-smooth behaviour of the performance is due to the uneven distributions of error among the objects.

bat and continued eliminating object pairs, according to the order of figure 5.1, up to the seahorse and the guitar. Figures 5.11 (a) and (b) plot recognition and pose estimation performance for each database and for different values of K.

Ideally, we would want the performance to be independent of the number of objects in the database. The experiment shows that although this is not the case, as we increase the database size from 7 to 21 objects, the performance of object recognition and pose estimation, for $k = 200$, drops by only 2.78% and 9.28%, respectively. This decrease in performance can be explained, to a certain extent, by the ambiguity that each object adds to the database. In general, each object has a number of views with little shape information due to the self-occlusion of parts from a given view. As we increase the number of objects, these views become less informative and start competing with each other. Consider, for example, the top-front view of the dinosaur, shown in Figure 5.4, in which its neck and head are self-occluded. As we add more four-legged objects, this view becomes more ambiguous. Hence, the results suggest that to maintain the performance under an increasing number of objects, we must not only explore strategies to reduce the number of redundant and ambiguous views in the database, but to add shape information to the index.

5.6 Occlusion by Missing Data

In the final experiment, shown in Figure 5.12, we plot recognition performance as a function of degree of occlusion (for the entire database) for occluded queries. To generate an occluded query, we randomly choose a node in the query DAG and delete the subgraph rooted at that node, provided that the order of the graph does not drop by more than 50%. We repeated the experiments 6 times, to accumulate a total of 16,128 trials. Performance decreases gradually as a function of occluder size (or, more accurately, the amount of "missing data"), reflecting the framework's ability to recognize partially visible

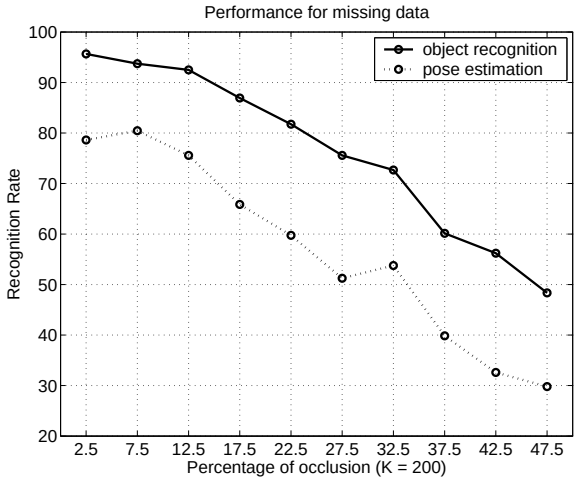


Figure 5.12: Recognition performance as a function of degree of occlusion, accumulated at fixed intervals of 5 degrees.

objects. Since an increasing amount of missing data does not have a dramatic impact on the performance, we can conclude that the framework satisfies this important property of stability. Experiments with other types of occlusion, in particular, under- and over-segmented shapes, are left as future work.

Chapter 6

Conclusions

Our shock labelling algorithm computes shock graphs that satisfy the necessary condition of structural stability required by the indexer and the matcher: shapes that suffer small deformations due to viewpoint changes are represented by shock graphs with similar structure. The stable labelling is obtained by first modelling the shock points in each skeleton branch by line segments. To this end, we proposed an efficient combinatorial algorithm that finds the minimum number of line segments that fits the data within a given threshold, which is set according to the number of points in the branch. We have also proposed a number of variations to the definition of shock graphs that lead to a more robust shape representation. In particular, we suggested that branch segments acting as ligature between shape parts should be encoded in single nodes[1]. This modification implies a small change to the shock graph grammar so as to allow chains of type 1 nodes. We believe that this simple modification has a large positive effect on the robustness of the graphs under viewpoint variations, where shape parts suffer large deformations due to perspective projection.

The above algorithm also suggested a node similarity measure with similar properties in terms of robustness to viewpoint changes. The key idea of this measure is to realise

[1] As we mentioned in Section 2.3.3, the original analysis of ligature nodes is due to August et al. In [2], the authors propose, in contrast to our approach, to eliminate the ligature nodes from the graph.

that the radius of shock points, and the parameters of their velocity and acceleration functions, yield a powerful feature that characterizes a node. Furthermore, by solely considering the nodes' radius function (i.e., effectively ignoring their absolute spatial information) we can gain a desirable flexibility in terms of deformation and articulation of parts, which is important to our task of object recognition.

Robust shock graphs are the right type of input for the indexing mechanism, since the indexing is based on accumulating votes for models with similar graph structure to that of the query. In turn, an optimal and efficient algorithm for counting votes is one of the contributions of the thesis. In the original formulation of the indexing, a query node was allowed to vote several times for different nodes in the same model. Conversely, a model's node was able to receive votes from more than one query node. In spite of the fact that an unconstrained assignment of votes from nearest-neighbour search results is likely to provide misleading information, it is sometimes tolerated due to the extra complexity that a bipartite-matching solution *per query-model pair*[2] would add to an algorithm that is designed, in principle, to provide a *fast* pruning of the database.

We solved the problem of an optimal accumulation of votes by an algorithm that adapts to the topology of the query graph and creates the minimum, fixed-size buffers that are necessary to compute an optimal one-to-one assignment of votes. Furthermore, the buffer size is independent of the number of models in the database, since it is determined by the number of overlapping signature vectors of the query with respect to a given range for the nearest-neighbour search. A second improvement to the discriminatory power of the indexer was obtained by using the signature vectors to reformulate the vote weighting function so as to favour models with more matched nodes and fewer unmatched nodes.

Our experiments show that the indexer is effective in reducing the number of candidates to be less than 10% of the model database. Furthermore, the matching performance

[2]Note that there is a potentially large number of models and so, when indexing we must avoid an expensive function that depends on this number.

is good even when the number of candidates is set to be 2% of the database. Since the indexing mechanism depends on the stability of the graphs under viewpoint changes, these results also suggest that shock graphs, computed as described in Section 2.5, successfully meet the viewpoint insensitivity condition.

Finally, we considered a number of extensions to the matching algorithm. The results of this algorithm originally suffered from inversions in the assignment of node correspondences that would also affect the overall similarity computed for a given pair of graphs —regardless of the particular domain. The inversions were due, in most cases, to the algorithm not being effective in guaranteeing that all the hierarchical constraints encoded in the DAGs were satisfied. Nevertheless, this algorithm had the advantage of a low complexity, $O(n^3)$, which was obtained, to a certain extent, by taking advantage of the signature vectors that were already computed for indexing. The goal was then to eliminate the problem of inversions while maintaining the low complexity of the algorithm. We proposed a method to update the similarity matrix at each iteration of the matching algorithm that not only eliminates the violation of hierarchical constraints, but also propagates information that promotes matching sibling nodes in the query to sibling nodes in the model. As shown in Section 4.4, this is important for structures (e.g., shapes) with repeated parts. In addition, a simple strategy was suggested for helping the algorithm proceed in a top-down fashion without compromising its ability to handle noisy graphs.

The results on object recognition and pose estimation show the matcher's ability to deal with non-isomorphic graphs, while the occlusion results confirm its robustness to increasing amounts of missing data. These properties are necessary conditions to handle shapes returned by a segmentation algorithm, which are likely to be either under- or over-segmented. Last but not least, the good performance results of these experiments validate the stability of the node similarity function to accommodate minor shape deformation due to viewpoint changes.

In conclusion, we have demonstrated an integrated framework for view-based 3-D

object recognition using shape information. The framework is effective in dealing with large databases of shapes without requiring a particular organization of its contents, such as prototypes, classes, or any other type of hierarchy. Hence, the indexing technique is a tool that can, in turn, be combined with different types of hierarchical databases. Both indexing and matching are general techniques applicable to domains where the patterns of interest can be embodied in stable, directed acyclic graphs. Finally, we have shown that our method for computing shock graphs satisfies this requirement of stability and, consequently, that shock graphs can be used for efficient shape matching.

6.1 Directions for Future Research

In order to effectively apply the framework to larger databases of objects, we must reduce the number of ambiguous views in the database, for as a database becomes larger, the negative effects of these views will become more noticeable. One form of view ambiguity is related to objects that look similar from different viewpoints, such as the case of the sphere, whose views are identical circles, or the case of a pawn (used in the experiments), whose side views are identical. That is, given one of these views, we cannot determine the exact viewpoint from where it was taken. Another form of ambiguity can be found in non-ambiguous views of an object that are shared by other objects in the database. The top view of the pawn and the lamp in Chapter 5, for instance, are undistinguishable.

By storing ambiguous views in the database, we increase the amount of redundant information, degrading the performance of the indexer, which will return many unnecessary candidates, and that of the matcher, which will have to evaluate and rank them. We can eliminate the former form of redundant views by grouping them according to their similarity and selecting a prototype to represent them all. Thus, in our framework, the similarity yielded by the matcher can be combined with a clustering technique to control the level of redundancy in the database. The later form of ambiguity (across objects)

requires that we explore other alternatives, such as assigning probabilities of occurrence to the views and to the objects [17]. In future work, we plan to evaluate the performance of the framework as a function of increasing numbers of objects using these ideas.

View clustering will reduce view ambiguity in the database. However, the indexing process introduces its own form of ambiguity through the use of the TSV. Recall that the TSV of a graph (subgraph) is not unique, and that the different graphs may share the same TSV. Moreover, the TSV encodes only the graph structure and not the shape information stored in the nodes. In Section 3.4.7, we considered a number of possible extensions to the indexer to overcome these issues. In particular, we suggested that large databases of objects will require signature vectors that also encode geometrical constraints. The geometrical constraints range from the individual labels and other attributes of the nodes to global properties, such as the relative position and size of the parts. This extra information can be encoded in the signature vectors as additional dimensions, or as edge weights in the graphs. Clearly, there are a number of features that we could consider at indexing time as well as a number of ways of doing so. The study of the signal-to-noise ratio of these different options is left as future work.

As a final point, we believe that the stability of shock graphs under articulation and occlusion of parts can be further extended by investigating the ligature segments and their role in the shock graph grammar. We envision a grammar that accounts for the process of part ligature so as to robustly represent articulating objects and the occlusion of parts. Ligature information can be used both as an indication of the birth process of parts, and as a measure of saliency for the nodes, for the information encoded in ligature nodes tends to be more unstable and less significant for matching. Shock graphs with well-identified parts and node saliency information will improve the performance of the matching process.

Bibliography

[1] C. Arcelli and G. S. di Baja. Ridge points in euclidean distance maps. In *Pattern Recognition Letters*, volume 4, pages 237–243, 1992.

[2] J. August, K. Siddiqi, and S. Zucker. Ligature instabilities in the perceptual organization of shape. *Computer Vision and Image*, 76(3):231–243, 1999.

[3] H. G. Barrow and R. M. Burstall. Subgraph isomorphism, matching relational structures, and maximal cliques. *Information Processing Letters*, 4(4):83–84, 1976.

[4] J. Beis and D. Lowe. Shape indexing using approximate nearest-neighbor search in highdimensional spaces. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1000–1006, 1997.

[5] I. Biederman. Recognition by components: a theory of human image understanding. *Psychol. Review*, 94:115–147, 1987.

[6] I. Biederman. Recognizing depth-rotated objects: A review of recent research and theory. *Spatial Vision*, 2000.

[7] I. Biederman and P. Gerhardstein. Recognizing depth-rotated objects: Evidence and conditions for three-dimensional viewpoint invariance. *Journal of Experimental Psychology: Human Perception and Performance*, 19(6):1162–1182, 1993.

[8] H. Blum. Biological shape and visual science. *Journal of Theoretical Biology*, 38:205–287, 1973.

[9] B.Luo and E.R.Hancock. A robust eigendecomposition framework for inexact graph matching. In *International Conference on Image Analysis and Processing*, pages 465–471, Palermo Italy, 2001. IEEE Computer Society Press.

[10] Christian Böhm, Stefan Berchtold, and Daniel A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys (CSUR)*, 33(3):322–373, 2001.

[11] Dragos Cvetkovic. *Spectra of Graphs: Theory and Applications*. Academic Press (San Diego), November 1997.

[12] Dragos Cvetkovic, Peter Rowlinson, and Slobodan Simic. *Eigenspaces of Graphs*. Cambridge University Press, January 1997.

[13] M. Cvetkovic D.M, Doob and H. Sachs. *Spectra of Graphs*. Academic Press, 1980.

[14] Christopher M. Cyr and Benjamin B. Kimia. 3d object recognition using shape similarity-based aspect graph. In *IEEE International Conference on Computer Vision*, pages 254–261, 2001.

[15] P. Dimitrov, C.Phillips, and K. Siddiqi. Robust and efficient skeletal graphs. In *IEEE Conference on Computer Vision and Pattern Recognition*, Hilton Head, SC, June 2000.

[16] Patrick J. Flynn and Anil K. Jain. 3d object recognition using invariant feature indexing of interpretation tables. *CVGIP: Image Understanding*, 55(2):119–129, 1992.

[17] W. T. Freeman. The generic viewpoint assumption in a framework for visual perception. *Nature*, 368:542–545, April 1994.

[18] H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for general graph matching problems. *Journal of the ACM*, 38:815–853, 1991.

[19] Christian Garcia-Arellano. Quantization techniques for similarity search in high-dimensional data spaces. Master's thesis, University of Toronto, 2002.

[20] Steven Gold and Anand Rangarajan. A graduated assignment algorithm for graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(4):377–388, 1996.

[21] J. A. Goldak, X. Yu, and A. K. Lingxian Dong. Constructing discrete medial axis of 3-d objects. In *Int. Journal of Computational Geometry and Applications*, volume 3, pages 327–339, 1991.

[22] Robert R. Goldberg and David G. Lowe. Verification of 3-d parametric models in 2-d image data. *Proc. of IEEE Workshop on Computer Vision*, pages 255–257, November 1987.

[23] J. Gomez and O. Faugeras. Reconciling distance functions and level sets. Technical Report Technical Report TR3666, INRIA, April 1999.

[24] A. Goralcikova and V. Konbek. A reduct and closure algorithm for graphs. *Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science 74:301–307, 1979.

[25] W.E.L. Grimson and T. Lozano-Pérez. Localizing overlapping parts by searching the interpretation tree. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 4(9):469–482, 1987.

[26] W. Hayward, M. Tarr, and A. Corderoy. Recognizing silhouettes and shaded images across depth rotations. *Perception*, 28:1197–1215, 1999.

[27] Gisli R. Hjaltason and Hanan Samet. Ranking in spatial databases. In *Symposium on Large Spatial Databases*, pages 83–95, 1995.

[28] J. Hopocroft and R. Karp. An n$\hat{2}$.5 algorithm for maximum matching in bipartite graphs. *SIAM Journal on Computing*, 2:225–231, 1973.

[29] C. Huang, O. Camps, and T. Kanungo. Object recognition using appearance-based parts and relations. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 877–883, San Juan, Puerto Rico, June 1997.

[30] C. Huang, O. Camps, and T. Kanungo. Hierarchical organization of appearance-based parts and relations for object recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, Santa Barbara, CA, June 1998.

[31] D. Huttenlocher and S. Ullman. Recognizing solid objects by alignment with an image. *International Journal of Computer Vision*, 2(5):195–212, 1990.

[32] A. Jepson and W. Richards. *What makes a good feature? in Spatial Vision in Humans and Robots (L. Harris and M. Jenkin Eds.)*, chapter 6. Cambridge University Press, 1993.

[33] Norio Katayama and Shin'ichi Satoh. The sr-tree: An index structure for high-dimensional nearest neighbor queries. In *ACM SIGMOD International Conference on Management of Data*, pages 369–380, May 1997.

[34] B. Kimia, A. Tannenbaum, and S. Zucker. Shapes, shocks, and deformations, i: The components of shape and the reaction-diffusion space. *International Journal of Computer Vision*, 15(3):189–224, 1995.

[35] Philip N. Klein, Thomas B. Sebastian, and Benjamin B. Kimia. Shape matching using edit-distance: an implementation. In *Symposium on Discrete Algorithms*, pages 781–790, 2001.

[36] D. Kozen. A clique problem equivalent to graph isomorphism. *SIGACT News*, pages 50–52, Summer 1978.

[37] Y. Lamdan, J. T. Schwartz, and H. J. Wolfson. On recognition of 3-d objects from 2-d images. *IEEE International Conference on Robotics and Automation*, pages 1407–1413, April 1988.

[38] F. Leymarie and M. D. Levine. Simulating the grassfire transform using an active contour model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(1):56–75, jan 1992.

[39] Y. Lovász and Pelicán. On the eigenvalues of a tree. *Periodica Math. Hung.*, 3:1082–1096, 1979.

[40] G. Malandain and S. Fernandez-Vidal. Euclidean skeletons. In *Image and Vision Computing*, volume 16, pages 317–327, 1998.

[41] D. Marr. *Vision*. Freeman, San Francisco, 1982.

[42] B. Messmer and H. Bunke. Subgraph isomorphism in polynomial time. Technical Report 95-003, IAM, 1995.

[43] John Midgley. Probabilistic eigenspace object recognition in the presence of occlusions. Master's thesis, University of Toronto, 2001.

[44] R. Ogniewicz. Automatic medial axis pruning by mapping characteristics of boundaries evolving under the euclidean geometric heat flow onto voronoi skeletons. Technical Report Technical Report 95-4, Harvard Robotics Laboratory, 1995.

[45] R.L. Ogniewicz. Skeleton-space: A multiscale shape description combining region and boundary information. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 746–751, Seattle, WA, June 1994.

[46] M. Pelillo, K. Siddiqi, and S. Zucker. Attributed tree matching and maximum weight cliques. *International Conference on Image Analysis and Processing*, September 1999.

[47] M. Pelillo, K. Siddiqi, and S. Zucker. Continuous-based heuristics for graph and tree isomorphisms. *Approximation and Complexity in Numerical Optimization: Continuous and Discrete Problems*, 1999.

[48] Steven W. Reyner. An analysis of a good algorithm for the subtree problem. *SIAM Journal on Computing*, 6(4):730–732, 1977.

[49] Thomas Sebastian, Philip Klein, and Benjamin Kimia. Recognition of shapes by editing shock graphs. In *IEEE International Conference on Computer Vision*, pages 755–762, 2001.

[50] Thomas B. Sebastian, Philip N. Klein, and Benjamin B. Kimia. Shock-based indexing into large shape databases. In *European Conference on Computer Vision*, pages 731–746, 2002.

[51] L.G. Shapiro. Structural shape description and matching,. In *PRIP79*, pages 413–420, 1979.

[52] A. Shokoufandeh and S. Dickinson. A unified framework for indexing and matching hierarchical shape structures. In *International Workshop on Visual Form*, pages 28–46, Capri, Italy, May 2001.

[53] A. Shokoufandeh, S. Dickinson, C. Jonsson, L. Bretzner, and T. Lindeberg. On the representation and matching of qualitative shape at multiple scales. In *European Conference on Computer Vision*, volume 3, pages 759–775, Copenhagen, Denmark, May 2002.

[54] A. Shokoufandeh, S. Dickinson, K. Siddiqi, and S. Zucker. Indexing using a spectral encoding of topological structure. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 491–497, Fort Collins, CO, June 1999.

[55] K. Siddiqi and B. B. Kimia. A shock grammar for recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, San Francisco, USA, 1996.

[56] K. Siddiqi, A. Shokoufandeh, Sven J. Dickinson, and Steven W. Zucker. Shock graphs and shape matching. In *IEEE International Conference on Computer Vision*, pages 222–229, 1998.

[57] S. Umeyama. An eigen decomposition approach to weighted graph matching problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(5):695–703, September 1988.

[58] Rakesh Verma and Steven W. Reyner. An analysis of a good algorithm for the subtree problem, corrected. *SIAM Journal on Computing*, 18(5):906–908, October 1989.