

Ma 113

ACTA POLYTECHNICA SCANDINAVICA

MATHEMATICS AND COMPUTING SERIES No. 113

Attribute Trees as Adaptive Object Models in Image Analysis

MARKUS PEURA

Helsinki University of Technology
Neural Networks Research Centre
P.O.Box 5400
FIN-02015 HUT, Finland

Dissertation for the degree of Doctor of Science in Technology to be presented with due permission of the Department of Computer Science and Engineering for public examination and debate in Auditorium T2 at Helsinki University of Technology (Espoo, Finland) on the 23rd of February, 2001, at 12 o'clock noon.

ESPOO 2001

Peura, M., **Attribute Trees as Adaptive Object Models in Image Analysis**. Acta Polytechnica Scandinavica, Mathematics and Computing Series No. 113, Espoo 2001, 80 pp. Published by the Finnish Academies of Technology. ISBN 951-666-567-5. ISSN 1456-9418. UDC 004.932:004.032.26:519.172

Keywords: Image analysis, trees, weighted trees, attribute trees, tree matching, unordered tree matching, heuristic tree matching, self-organizing map.

ABSTRACT

This thesis focuses on the analysis of irregular hierarchical visual objects. The main approach involves modelling the objects as unordered attribute trees. A tree presents the overall organization, or topology, of an object, while the vertex attributes describe further visual properties such as intensity, color, or size. Techniques for extracting, matching, comparing, and interpolating attribute trees are presented, principally aiming at systems that can learn to recognize objects. Analysis of weather radar images has been the pilot application for this study, but the main ideas are applicable in structural pattern recognition more generally.

The central original contribution of this thesis is the Self-Organizing Map of Attribute Trees (SOM-AT) which demonstrates how the proposed tree processing techniques — tree indexing, matching, distance functions, and mixtures — can be embedded in a learning system; the SOM-AT is a variant of the Self-Organizing Map (SOM), the neural network model invented by Prof. Teuvo Kohonen. The SOM is especially suited to visualizing distributions of objects, classifying objects and monitoring changes in objects. Hence, the SOM-AT can be applied in the respective tasks involving hierarchical objects. More generally, the proposed ideas are applicable in learning systems involving comparisons and interpolations of trees.

The suggested heuristic index-based tree matching scheme is another independent contribution. The basic idea of the heuristic is to divide trees to subtrees and match the subtrees recursively by means of topological indices. Given two attribute trees, the larger of which has N vertices, and the maximal child count (out-degree) is \overline{D} vertices, the complexity of the scheme is only $\mathcal{O}(N \log \overline{D})$ operations while exact matching schemes seem to have at least quadratic complexity: $\mathcal{O}(N^{2.5})$ operations in checking isomorphisms and $\mathcal{O}(N^3)$ operations in matching attribute trees. The proposed scheme is efficient also in terms of its “hit rate”, the number of successfully matched vertices. In matching two random trees of $N \leq 10$ vertices, the number of heuristically matched vertices is on average 99% of the optimum, and the accuracy for trees of $N \leq 500$ vertices is still above 90%.

The feasibility of the proposed techniques is demonstrated by database querying, monitoring, and clustering of weather radar images. A related tracking scheme is outlined as well. In addition to weather radar images, a case study is presented on Northern light (*Aurora Borealis*) images. Due to the generic approach, there are also medical and geographical applications in view.

© All rights reserved. No part of the publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author.

Preface

In September 1996, I had an interesting problem at hand, as well as research facilities, funding, and some optimism.

For the interesting problem — analysis of weather radar images — as well as for related guidance and support, I thank Jarmo Koistinen, Elena Saltikoff and Robin King at the Finnish Meteorological Institute. I am also grateful to Mikko Syrjäsuo at the Geophysical Research of the FMI for cooperation in the study on auroral images.

For the research facilities, I am grateful to Academy Professor Erkki Oja, the current head of the Neural Networks Research Centre. I thank him also for his encouraging supervision in this rather independent work. In fact, the objects I have dealt with — graphs — are something he explored years ago but he kindly consented to see them on his desk again. I appreciate his flexibility and also his straightforward style in steering the Lab. For research facilities and scientific support, I am also grateful to Academician Teuvo Kohonen, the founder of the Centre, whose world success with Self-Organizing Maps has meant success for the whole Lab. This thesis is, to a large extent, yet another piece of science among thousands of other reported works based on the SOM. In 1996-1997, I spent an academic year at Université Louis Pasteur, Strasbourg; I thank Professor Jerzy Korczak, the head of the Artificial Intelligence Group, for local research facilities and support.

For the funding, I express my thanks to the Lab itself, as well as to HeCSE, Helsinki Graduate School in Computer Science and Engineering. For grants, I am grateful to the Foundation of Technology, Research Foundation of Helsinki University of Technology, and Finnish Academy of Science and Letters.

For optimism, one needs friends and inspiring colleagues. I thank all my colleagues here at the Lab for so many interesting informal and formal discussions, as well as all the friends for support. I am especially obliged to Harri, Esa, Jaro, Patrik, Jampe, Xavier, Sami, Jaakko S., Panu, Vuokko, Krista, Lea, Kimple, Jorma, Tove, Jone, and Turku Sauna Boys — for help, comments, and activities. I express my thanks also to my friends in Strasbourg — Kathy, Caroline, David, Jean-Pierre, Nicolas, and José. The year in Strasbourg was indeed inspiring; I guess that I got the majority of the ideas related to this thesis there — it however took me these years to wrap them up. Later on, my engagement with trees snatched me in the jungle of computational complexities. Therefore, I thank Professor Ilkka Niemelä, L.Sc. Harri Haanpää, and L.Sc. Tommi Junttila in the Laboratory for Theoretical Computer Science for many discussions. I express my special thanks to Professor Heikki Mannila for help as well as to Professor Pekka Kilpeläinen, reviewer of this thesis, for comments as well as for outlining a stricter bound of complexity for my matching scheme. I thank also the second reviewer, Professor Heikki Kälviäinen, for helping me in seeing the essential.

Finally, I send my warmest thanks to my wife Liisa and our daughter Telma, who have supported me and understood my clouds and trees.

Otaniemi, February 8, 2001

Markus Peura

Contents

Preface	3
List of notations	7
List of publications	8
1 Introduction	11
2 Tree matching and related problems	13
2.1 Definitions	13
2.2 Basic matching problems	14
2.3 Graph problems	17
2.4 Tree problems	18
2.5 Syntactic tree recognition	24
3 Heuristic approaches	25
3.1 General	25
3.2 Heuristics for matching problems	26
4 Proposed approach: index-based matching	28
4.1 Basic idea	28
4.2 Indices	30
4.3 Weighted bipartite matching of indices	32
4.4 Overall complexity	33
5 Trees as object representations in learning systems	34
5.1 The role of weights	34
5.2 Merging	35
5.3 Consistency of matching and merging	37
5.4 The self-organizing map of attribute trees	44
6 Bounds of matching performance	49
6.1 Indexing, index-based matching or exact matching?	49
6.2 Lower bounds for matching — performance of random indices	52
6.3 Experimental performance of the proposed indices	56

7	Computer vision applications	60
7.1	Spatial hierarchy and trees	60
7.2	Attributes	61
7.3	Weather radar images	62
7.4	Auroral images	65
7.5	Known problems	66
7.6	Future applications	67
8	Conclusions	68
	References	69
A	Probabilities and expectations related to coinflip trees	76
A.1	Expectation minimum child count	76
A.2	Derivation of $P(n)$ and $P(n \overline{H}, \overline{D})$	77
A.3	Sample data set	79

List of notations

Graphs and trees

$G = (V, E)$	graph
$G = (U, V, E)$	bipartite graph
$T = (V, E)$	tree
$T = (V, E, w)$	weighted tree
$T = (V, E, \mathbf{a})$	attribute tree
$T = (V, E, w, \mathbf{a})$	weighted attribute tree
u, v	vertex (node, point)
λ	null vertex, nonexistent vertex
U, V	set of vertices
$e, (v_i, v_j), (u, v)$	edge (link, arc, line)
E	set of edges
w	weight, a mapping from vertices to weights
$w(v_i), w_i$	weight of v_i
\mathbf{a}	attribute vector, a mapping from vertices to attribute vectors
$\mathbf{a}(v_i), \mathbf{a}_i$	attribute vector of v_i
$a(v_i), a_i$	attribute of v_i
$\text{parent}(v)$	parent of v
$V^+, V^+(v)$	children of v
v_i	child of v
v_i^*	descendant of v
T_v	tree, the vertex set of which is denoted by V
$T(v)$	tree rooted at v
$T(v_i), T_i$	tree rooted at v_i

Tree indexing

I	index (descriptor, property, feature, invariant, fingerprint) of a tree (complete list in Table 4, p. 30)
$I(v), I_v, I$	index of $T(v)$
$I(v_i), I_i$	index of $T(v_i)$
\tilde{I}	weighted version of index I
\bar{I}	maximum of index I
$ V , N$	number of vertices in a tree
D	number of children (out-degree) of a vertex
H	height of a tree
Δ	depth of a vertex (path length from root)
\mathbf{i}	index vector
$\mathbf{i}(v), \mathbf{v}$	index vector of $T(v)$

Tree matching (assume T_u and T_v to be matched)

$T_m = (M, M', E_m)$	tree describing a matching of T_u and T_v
M, M'	vertex sets of T_m
m	vertex in M , identified with a pair of matched vertices: $m = (u, v)$ for some $u \in T_u$ and $v \in T_v$
m'	vertex in M' , identified with an unmatched vertex: $m' = u$ or $m' = v$ for some $u \in T_u$ or $v \in T_v$
E_m	edge set of T_m
$c(u, v)$	cost of matching u and v
$c'(v)$	cost of leaving v unmatched
$\hat{c}(u, v)$	approximated, unscaled cost of matching $T(u)$ and $T(v)$, based on the respective index vectors \mathbf{u} and \mathbf{v}
$\hat{c}'(v)$	approximated, unscaled cost of leaving $T(v)$ unmatched, based on the respective index vector \mathbf{v}
$\text{cost}(T_u)$	cost of leaving T_u unmatched
$\text{cost}(T_m)$	cost of matching T_u and T_v (implicitly presented by T_m)
$\text{dist}(T_u, T_v)$	distance between T_u and T_v
$\widehat{\text{dist}}(T_u, T_v)$	approximated distance, by default equal to $\text{cost}(T_m)$ calculated from a heuristically obtained matching T_m

Other

$\mathcal{O}(f(n))$	computational complexity at most of order $f(n)$
P, NP	classes computational complexity (see Def. 5)
PROBLEM	computational problem
program, program(...),	computer program
inside, adjacent-to	spatial relation

Unless stated otherwise, the trees discussed in this thesis are assumed to be rooted and unordered.

List of publications

- I Jukka Iivarinen, Markus Peura, Jaakko Särelä, and Ari Visa. Comparison of combined shape descriptors for irregular objects. In Adrian F. Clark, editor, *8th British Machine Vision Conference (BMVC'97)*, volume 2, pages 430–439, September 1997.
- II Markus Peura. A statistical classification method for hierarchical irregular objects. In Alberto del Bimbo, editor, *Image Analysis and Processing — 9th International Conference, (ICIAP'97)*, volume 1310 of *Lecture Notes in Computer Science*, pages 604–611. Springer, September 1997.
- III Markus Peura. The self-organizing map of trees. *Neural Processing Letters*, 8(2):155–162, October 1998.
- IV Markus Peura, Elena Saltikoff, and Mikko Syrjäsoo. Image analysis by means of attribute trees — remote sensing applications. In *IEEE 99 International Geoscience and Remote Sensing Symposium (IGARSS'99)*, pages 696–698, June–July 1999.
- V Markus Peura. Classifying, monitoring and tracking precipitation cells by means of attribute trees. In *29th International Conference on Radar Meteorology*, pages 86–89. American Meteorological Society, July 1999.
- VI Markus Peura. The self-organizing map of attribute trees. In *International Conference on Artificial Neural Networks (ICANN99)*, pages 168–173. IEE, September 1999.
- VII Markus Peura. Attribute trees in image analysis — heuristic matching and learning techniques. In *10th International Conference on Image Analysis and Processing (ICIAP'99)*, pages 1160–1165. IEEE Computer Society, September 1999.

Summary of the publications

In Publ. I, efficiency of shape descriptors was illustrated by means of the Self-Organizing Map (Kohonen 1990). The descriptors studied were the *chain code histogram* (Iivarinen and Visa 1996), the *pairwise geometric histogram* (Ashbrook et al. 1995) and a set of simple, global indices (*convexity, elongation, compactness, circularity, ellipticity*). The author's contribution was in designing and experimenting the simple shape indices together with Iivarinen; the workshop paper of Peura and Iivarinen (1997) presents related issues in further detail. The underlying application was classification of paper defects. However, the studied descriptors appear to be generally applicable to irregular shapes. Hence, with respect to this thesis, the role of Publ. I is in presenting attributes to be used in the trees discussed in the other publications.

In Publ. II, a simple method for indexing trees is presented and tested with artificial visual objects. The basic idea is to calculate statistics of topology and shapes at first at each level and then vertically, over the levels. The approach is simple but can be regarded as an introduction to the recursive, tree-matching related indexing proposed in the later publications. Publ. II also contains the first version of the TRIM algorithm for extracting trees from images.

Publ. III shows how rooted unordered trees can be used as the learning medium of the Self-Organizing Map (SOM; Kohonen 1990); the original and most popular version of SOM uses vectors. While the SOM learning rules are essentially general, there is however no straightforward scheme for adapting them to arbitrary data structures. The major contribution of Publ. III is the definition of the learning rules for trees by an index-based tree matching scheme. The promising results with the SOM suggest that the proposed techniques have general applicability in arbitrary interpolative learning systems, too. Publ. III is rather independent from Publications I and II, as no special assumptions are made on applications.

Publ. IV introduces briefly two remote sensing applications of tree matching. The author is responsible for the most part of this publication. E. Saltikoff and M. Syrjäsuo participated in problem formulation, provided the involved data and background expertise. The discussion on analysis of weather radar images is based on a more detailed seminar paper by Peura et al. (1998).

Publ. V presents three meteorological applications of tree matching in analysis of radar images. Instead of elaborating technical details, the goal of is to transfer ideas in a general level to the meteorological community.

Publ. VI generalizes the topological approach of Publ. III to the Self-Organizing Map of Attribute Trees. Radar image data is applied in the experiment.

Publ. VII is the synthesis of the topics listed this far, binding together the TRIM algorithm, tree matching, mixing, the SOM of Attribute Trees, radar and auroral images.

1 Introduction

This work belongs to the field of pattern recognition, introducing new techniques in modelling objects as trees. The emphasis is on adaptive recognition systems incorporating tree matching in one form or another. In applications, the main scope is on computer vision but the developed methods are applicable in pattern recognition and related fields more generally. The main assumption is that the objects are somehow hierarchical and hence suited to tree-structured modelling. On the other hand, the trees are allowed to be noisy, irregular, complex, large, and dynamical in size and form.

Formally, a tree is a connected acyclic graph. The trees discussed in this thesis are assumed to be rooted, unordered and allowed to contain weights or other attributes. The unorderedness means that the order of vertices connected to a parent vertex is not taken into account. The assumption of unorderedness is justified for example in many computer vision applications in which the spatial left-right and top-bottom order of objects is insignificant. Ordered trees, applied for example in string matching, are beyond the scope of this thesis.

Tree matching and *tree interpolation* are central topics of this thesis. Tree matching is applied for example in comparing trees and thus in evaluating *distances* between trees, which is essential in applications involving recognition and classification. Tree interpolation means computing “weighted sums” of trees; the obtained mixtures can be further interpreted as smooth generalizations of the involved trees. In the context of classification, generalizations provide a means for presenting and storing knowledge compactly. Trees can be also *monitored* and *tracked* by means of tree matching: dynamics of a phenomenon can be analyzed by studying changes in the respective series of trees.

The principal contribution of this thesis is the *Self-Organizing Map of Attribute Trees* (SOM-AT), a variant of the *Self-Organizing Map* (SOM), the neural network model invented by Kohonen (1982, 1990, 1995). The SOM-AT demonstrates how tree matching and interpolation can be incorporated in a learning system such that the core of the system, the learning medium, consists of trees. The approach differs from standard continuous learning paradigms in which the typical choice is to use vectors as a learning medium — practically implying that the studied objects as well must be presented as vectors.

Research on tree matching has been carried out under various problem formulations. This thesis concentrates on `FIXED TREE MATCHING`, a problem slightly harder than `SUB-TREE ISOMORPHISM` (Matula 1968, Reyner 1977, Pelillo et al. 1999b); the respective time complexities of these problems are $\mathcal{O}(N^3)$ and $\mathcal{O}(N^{2.5})$, where N is the number of vertices. The less constrained problem `TREE EDIT DISTANCE` (or `EDITED TREE MATCHING`) is remarkably harder, **NP**-complete (Zhang et al. 1992, Shasha et al. 1994). Due to the complexity, this problem has less emphasis in this thesis, but one may expect that for instance the proposed interpolation techniques can be adapted to this problem, too. On the other hand, matching is not the only technique for recognizing trees. Trees can be

modelled syntactically, and then classified and generalized using related techniques (Fu and Bhargava 1973). However, one of the principal assumptions in this work is that the objects of interest do not necessarily contain repeated structures, making syntactic approaches less feasible.

In general, learning processes involve massive iterations. Consequently, the involved low-level operations, such as distance computations and adaptation steps, should be computationally light, say less than quadratic, $\mathcal{O}(N^2)$. The proposed *heuristic scheme for approximating* FIXED TREE MATCHING is the second independent contribution of this thesis. The heuristic is generally applicable — besides being especially suited to the SOM-AT. The basic idea is to divide a tree recursively into subtrees, and apply *indices*, numerical characteristics of the subtrees, in matching instead of exhaustive trials. The complexity of the scheme is practically linear, $\mathcal{O}(N \log \bar{D})$, where N is the vertex count of the larger tree and \bar{D} is the maximum child count. The general scheme of using the proposed approach is shown in Fig. 1.

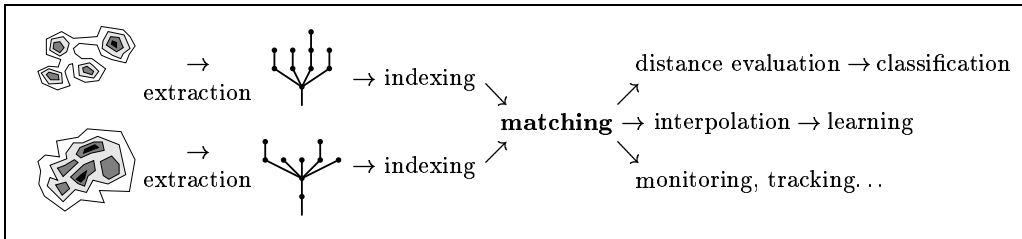


Figure 1: Overall processing scheme for trees.

The third contribution is to use trees as object models in the *analysis of weather radar images*. Precipitative clouds, and clouds in general, are a phenomenon typically containing irregular spatial hierarchies. By spatial hierarchy, we mean the topological organization defined by the nesting and adjacency of image segments. In this context, irregularity means that the objects of interest do not have to be syntactic or geometric. The images provided by the Finnish Meteorological Institute have inspired many of the ideas presented in this thesis and served as a primary testbench for the related algorithms.

This introductory part of the thesis is organized as follows. General definitions and matching problems are presented in Sec. 2. The related complexities are included as a motivation for heuristical approaches. Some heuristics for matching problems are reviewed in Sec. 3, followed by a synthesis of the proposed techniques for tree matching in Sec. 4. In Sec. 5, the techniques are further developed towards the applicability in learning systems, with a special emphasis on the Self-Organizing Map of Attribute Trees. Also the consistency of continuous tree interpolations — a topic not addressed in the publications — is discussed in detail. The matching performance of the proposed scheme discussed in Sec. 6 is another previously unpublished topic. Practical applications related to image analysis and potential future research are discussed in Sec. 7. Finally, conclusions are presented in Sec. 8.

2 Tree matching and related problems

2.1 Definitions

Definition 1 (Distance). A *distance function* or a *metric* is a function satisfying the following axioms (Santini and Jain 1999):

$$\begin{aligned}
 \text{dist}(A, A) &= \text{dist}(B, B) && \textit{(self-similarity)} \\
 \text{dist}(A, B) &\geq \text{dist}(A, A) && \textit{(minimality)} \\
 \text{dist}(A, B) &= \text{dist}(B, A) && \textit{(symmetry)} \\
 \text{dist}(A, B) + \text{dist}(B, C) &\geq \text{dist}(A, C) && \textit{(triangle inequality)}
 \end{aligned} \tag{1}$$

Instead of exact distances we will often address distance approximations, denoted by $\widehat{\text{dist}}(A, B)$, for which the above axioms are not expected to hold strictly.

Definition 2 (L -norm). Given a vector $\mathbf{x} = [x_1 \ x_2 \ \dots \ x_n]^\top$, its L_p -norm is

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^n |x_i|^p \right)^{\frac{1}{p}} \tag{2}$$

where $p > 0$. If discussion applies to any $p > 0$, we will refer to an L -norm generally and write $\|\mathbf{x}\|$ without subscript. L -norm of a vector difference $\|\mathbf{x} - \mathbf{y}\|$ is called the *Minkowski distance* between \mathbf{x} and \mathbf{y} (Devijver and Kittler 1982).

Definition 3 (Graphs). A *graph* $G = (V, E)$ consists of a set of vertices $V = \{v_i\}$ and edges $E = \{e_i\}$. The edges are connections between vertices; vertex v_j is *adjacent* to v_i if there is an edge $e = (v_i, v_j)$ between them. Two edges are adjacent if they have a common vertex. In an *undirected* graph $(v_i, v_j) \equiv (v_j, v_i)$. A *path* is a chain of adjacent edges $(v_{(1)}, v_{(2)}), (v_{(2)}, v_{(3)}), (v_{(3)}, v_{(4)}), \dots, (v_{(k-1)}, v_{(k)}) \equiv \text{path}(v_{(1)}, v_{(k)})$, such that no vertex is visited more than once except for the starting vertex $v_{(1)}$, which may be same as the ending vertex $v_{(k)}$. If $v_{(1)} = v_{(k)}$, the path is a *cycle*. If there is a path between any two vertices, the graph is *connected*.

Definition 4 (Trees). A *tree* $T = (V, E)$ is an undirected connected graph that contains no cycles. In a *rooted tree*, one of the vertices is distinguished as the *root*, r . Along any path starting from the root, the adjacent vertices $v_{(i)}$ and $v_{(i+1)}$ are called a *parent* and a *child*, respectively, and denoted $v_{(i)} = \text{parent}(v_{(i+1)})$. Thus, except for the root, every vertex has a unique parent but may have several children or none. The children of a common parent are *siblings*. If the order of the siblings is distinguished the tree is *ordered*, otherwise *unordered*. The children of v , their children, and so on are the *descendants* of v , denoted by v_i^* . The *subtree* rooted at v contains hence v and its descendants. In an *attribute tree* $T = (V, E, \mathbf{a})$ each vertex is associated with an attribute or several attributes, an attribute vector: $\mathbf{a}(v_i) = \mathbf{a}_i$. In this thesis, all the attributes are assumed to be numeric. The *weight* $w(v_i) = w_i$ of a vertex is a special attribute. By default,

$w_i \in [0, 1]$. At this point one may consider the weight as a degree of significance, strength or existence of a vertex; the role of weight will be discussed in further detail in Sec. 4. A *weighted tree* (having no other attributes) is a triplet $T = (V, E, w)$. Finally, a *weighted attribute tree* is denoted by $T = (V, E, w, \mathbf{a})$. In a rooted tree, there is no reason to distinguish between the attributes of edges and vertices since every edge is associated with a unique vertex — the child vertex where the edge points to. Any property of an edge can be seen as a property of that vertex.

Definition 5 (Complexity). In this thesis, computational effort means *computation time* defined as the number of computation steps with respect to the length of the input in solving a problem (exactly or approximately). We say that a problem having input length n is of order $f(n)$, denoted $\mathcal{O}(f(n))$, if the number of operations implied by the input is at most $cf(n)$ for all inputs of length $n > N$ for some fixed N and c . *Computational complexity* refers to the computational effort defined through the $\mathcal{O}(f(n))$ notation. Omitting a bunch of theoretical constructions based on the concept of *Turing machines*, we mention that there are two classes of special interest in computer science. The class of all the polynomial-time problems that can be solved by a deterministic computation is denoted by \mathbf{P} . Accordingly, \mathbf{NP} stands for the class of problems that can be solved in a polynomial time by a nondeterministic computation; nondeterminism means the ability to guess the correct computation paths. A nondeterministic solution to a problem means that there *exists* at least one computation path leading to the solution. A practical property is that any problem in \mathbf{NP} can be solved in exponential time. \mathbf{P} is a subset of \mathbf{NP} , it is thus conventional to say “a problem is in \mathbf{NP} ” when actually meaning that the problem is known to be outside \mathbf{P} but still inside \mathbf{NP} . This thesis will not address problems beyond \mathbf{NP} .

Definition 6 (Reduction). A *reduction* is a process of reformulating a problem as another problem such that the solution of the former problem can be obtained by solving the latter. Assuming that the complexity of the latter problem is known, the reduction is an indirect tool for determining the (maximal) complexity of the former problem. A problem is said to be *NP-complete* if any problem in \mathbf{NP} can be reduced to it; the further constraint for the reduction in this case is that it can be computed in polynomial time (Garey and Johnson 1979) or in logarithmic space as preferred by Papadimitriou (1994).

2.2 Basic matching problems

Tree matching is a special case of more general matching problems. In order to discuss tree matching problems, some basic matching problems are reviewed first.

In established terminology of graph theory, a *matching* is generally defined as a subset of the edges of a given graph $G = (V, E)$ such that no two edges are adjacent. Sometimes, matching is defined more specifically between two distinct vertex sets $U = \{u_i\}$ and $V = \{v_j\}$, thus assuming a *bipartite graph* $G = (U, V, E)$ where $E \subseteq U \times V$. For the time

being, let us at first define a matching generally as any subset of the edges in a bipartite graph, without constraints on adjacency:

$$M = \{m_i\} \subseteq E. \quad (3)$$

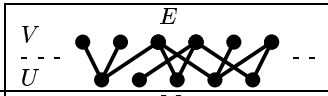
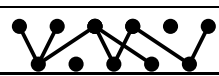

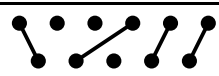
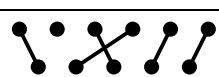


According to this point of view, E can be regarded as a list of permitted connections. Further, let us denote the set of *unmatched vertices* by

$$M' = \{m' \mid m' \in U, \nexists v : (m', v) \in M\} \cup \{m' \mid m' \in V \nexists u : (u, m') \in M\}. \quad (4)$$

Matching principles. Clearly, there are several ways of matching the vertices of U to those of V . Some matching principles are listed in Table 1. We say that matching is *maximal* (under a principle) if $|M|$ maximizes. The matching principle depends on the practical application at hand. Typically, determining a matching principle for an application is not straightforward. It must be noted that the principle dictates an essential part of the nature of the approach, compromising between tractable computational effort and satisfactory results.

Some classical matching problems related to the principles listed in Table 1 are of special interest here. In the classical combinatorial problem of BIPARTITE MATCHING one tries to find a maximal one-to-one matching. Sometimes this problem is called PERSONNEL ASSIGNMENT PROBLEM, since it can be thought as of assigning employees $u_i \in U$ to tasks $v_j \in V$ (Bondy and Murty 1976). Hopcroft and Karp (1973) presented an exact algorithm having complexity of $\mathcal{O}(N^{\frac{5}{2}})$.

Table 1: Matching principles for a bipartite graph $G = (U, V, E)$.

matching principle	constraint	
many-to-many		
complete many-to-many	$[\forall u \exists v : (u, v) \in M] \wedge$ $[\forall v \exists u : (u, v) \in M]$	
one-to-one	$\forall (u_i, v_j) \in M, \forall (u_k, v_l) \in M :$ $(i = k) \Leftrightarrow (j = l)$	
complete one-to-one	one-to-one such that $[\forall u \exists v : (u, v) \in M] \vee$ $[\forall v \exists u : (u, v) \in M]$	
one-to-many	$\nexists (i, j, l) : (i \neq l) \wedge$ $(u_i, v_j), (u_i, v_l) \in M$	
mixed disjoint	$\nexists (i, j, k, l) : (i \neq k) \wedge (j \neq l) \wedge$ $(u_i, v_j), (u_k, v_j), (u_k, v_l) \in M$	

Numerical constraints. In addition to the constraints listed in Table 1, a matching problem often involves minimization of a cost function of the form

$$\text{cost}(M, M') = \sum_{m \in M} c(m) + \sum_{m' \in M'} c'(m'). \quad (5)$$

where $c(m)$ is the cost of matching u to v , that is, assigning edge $m = (u, v)$ in M , and $c'(m')$ is the cost of leaving a vertex unmatched ($m' = u$ or $m' = v$). For clarity, we may write $c(u, v)$ instead of $c(m)$ and respectively $c'(u)$ or $c'(v)$ instead of $c'(m')$. The c and c' can be regarded as *repulsion* and *attraction* in physics. Using (5), BIPARTITE MATCHING can be formulated by setting, say, $c = 3$ for all edges and $c' = 2$ for unmatched vertices, which implies maximal 1-to-1 matching: it is profitable to match any pair of yet unmatched vertices ($3 < 2+2$) but not profitable to reuse a matched vertex for catching a single unmatched vertex ($3 > 2$).

The formulation of cost functions in (5) depends on the application. It is often practical to assume that each vertex is associated with an attribute or an attribute vector, that is, a point in space. The cost of matching vertices u and v can then be defined, say, as the Euclidean distance between the respective attribute vectors \mathbf{u} and \mathbf{v} . One may further assume that in, addition to attributes, vertex *weights* are applied. In this thesis, a “weight” refers to a vertex weight, unless stated otherwise. A weight describes the strength, activity, probability or significance of a vertex. By default, the weight is $w = 1$, and a vertex with $w = 0$ can be treated as a nonexistent vertex. For the cost function (5) it is hence natural to assume that:

1. $c'(u)$ is monotonically increasing with respect to w_u .
2. $c(u, v)$ is monotonically increasing with respect to w_u, w_v , and $\|\mathbf{u} - \mathbf{v}\|$.
3. Matching is favored over unmatching: $c(u, v) \leq c'(u) + c'(v)$.

It is often sensible for c' to regard unmatched vertices as virtually matched against “null vertices” having zero-valued weights or attribute vectors. Suggestions of cost functions c and c' are listed in Table 2.

Table 2: Definitions of vertex matching and unmatching costs. The attributes of vertices u and v are denoted by \mathbf{u} and \mathbf{v} and weights by w_u and w_v , respectively.

	$c(u, v)$	$c'(u)$
default; no weights or attributes	0	1
weighted vertices	$ w_u - w_v $	$ w_u $
attributed vertices	$\ \mathbf{u} - \mathbf{v}\ $	$\ \mathbf{u}\ $
weighted, attributed vertices	$\frac{w_u + w_v}{2} \ \mathbf{u} - \mathbf{v}\ $	$w_u \ \mathbf{u}\ $

In WEIGHTED BIPARTITE MATCHING, each edge (u, v) is associated with a profit $p(u, v)$, and one tries to find a complete one-to-one matching that maximizes the sum of profits. (In literature, profit is often called weights, a term which would be misleading in

this context.) Again, one may think of employees, now having varying effectiveness in different tasks; hence the synonym OPTIMAL ASSIGNMENT PROBLEM (Bondy and Murty 1976, Lawler 1976). The problem can be defined equivalently through (5) with, say, $c'(v) = 1$ and $c(u, v) = 2 - p(u, v)/[\max(p) - \min(p) + \epsilon]$. The computational complexity appears currently (Buss and Yianilos 1998) to be $\mathcal{O}(N^3)$ in general case (Lawler 1976) and $\mathcal{O}(N^{\frac{5}{2}} \log N)$ for profits (weights) defined as L-norms (Def. 2) in a plane (Vaidya 1989).

Remark. BIPARTITE MATCHING can be *reduced* (see Def. 6) to a more general graph problem called MAXIMUM FLOW, which can be likewise solved in polynomial time (Papadimitriou 1994). In that problem, one tries to maximize the flow through a network in which each (directed) edge has a predefined integer capacity. BIPARTITE MATCHING can be embedded in a MAXIMUM FLOW formulation regarding each $e \in E$ as a channel of unit capacity. If and only if the flow of magnitude N can be directed from U to V , there exists a complete one-to-one matching. Clearly, also WEIGHTED BIPARTITE MATCHING can be reduced to MAXIMUM FLOW. The findings of the future research on MAXIMUM FLOW may yield even smaller upper bounds for BIPARTITE MATCHING and WEIGHTED BIPARTITE MATCHING. However, although the revised algorithms are progressively simpler in terms of theoretical computational complexity, their implementations tend to be practically less attractive due to numerous additional data structures and gadgets. For example, see the algorithm of Ahuja et al. (1989) having complexity $\mathcal{O}(NM \log((N/M)(\log c)^{1/2} + 2))$, where $N = |V|$, $M = |E|$ and c is the maximal capacity of flow.

Finally, it is natural to define the *distance* between U and V as

$$\text{dist}(U, V) = \min_M [\text{cost}(M, M')]. \quad (6)$$

The M corresponding to the distance is an *optimal matching*. Unless stated otherwise, cost is defined according to (5) and Table 2.

2.3 Graph problems

In the above discussion, a matching was defined *within* a bipartite graph, between the two separate vertex sets which by definition have no internal structure. On the other hand, if two vertex sets do have internal structure to be taken into account in matching, the problem is called *graph matching*. In other words, matching is constrained not only by the vertex-to-vertex costs of type (5) but also by the preservation of the topology, that is, neighborhood relations of the vertices. Three purely topological graph problems closely related to this thesis are shown in Fig. 2.

Two graphs are *isomorphic* if they can be obtained from each other by relabeling vertices. GRAPH ISOMORPHISM is hence the problem of finding a bijective mapping $f : U \rightarrow V$ between two graphs $G_u = (U, E_u)$ and $G_v = (V, E_v)$ such that $(f(u), f(u')) \in E_v$ if and only if $(u, u') \in E_u$. It is one of the classical problems in computer science hav-

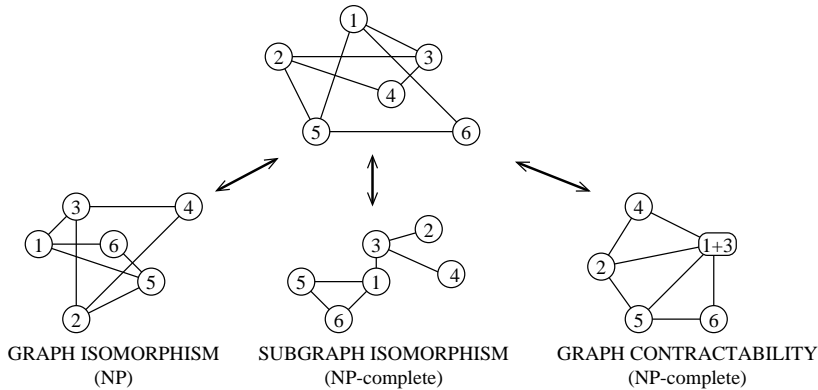


Figure 2: Graph matching problems. The labels illustrate possible solutions.

ing applications for example in pattern recognition. The problem remains inside **NP**; the brute force solution requires testing all the permutations of the $|U| = |V|$ vertices. However, determining the **NP**-completeness of **GRAPH ISOMORPHISM** has remained one of the yet unresolved challenges (Papadimitriou 1994). In checking the isomorphism of many graphs against a single graph, some speedup can be obtained using enumeration approach: instead of comparing two complex structures (graphs) directly to each other, one calculates and compares their *ranks*, that is, integers uniquely associated with topologies (Kreher and Stinson 1999).

Practical algorithms, engaging search trees in a form or another, can be optimized by pruning unnecessary branches of the search trees. Such technique has been used by Ullmann (1976) with the problem of **SUBGRAPH ISOMORPHISM**, where one searches for a subgraph of G that is isomorphic with G' . Computational effort for worst cases remains nevertheless exponential; **SUBGRAPH ISOMORPHISM** is **NP**-complete (Garey and Johnson 1979).

Detection of isomorphisms is often too strict an approach for natural objects. In pattern recognition problems, one typically wishes to determine also a degree of similarity in terms of the number of *edit operations* required in deforming a graph to another. In **GRAPH CONTRACTABILITY** (Garey and Johnson 1979), one checks whether a graph G' can be obtained from G by repeatedly merging pairs of vertices of G such that each compound vertex inherits the edges of the original vertices. Also this problem is **NP**-complete.

2.4 Tree problems

Tree problems corresponding to the graph problems of Fig. 2 are shown in Fig. 3. As trees are a subclass of graphs, one could assume that the corresponding problems be easier. It appears that some problems indeed are, falling to **P**, while others persist in

NP. A list of complexities of both ordered and unordered tree problems was published by Kilpeläinen and Mannila (1994). In the following, some fundamental tree matching problems with respect to this thesis are reviewed.

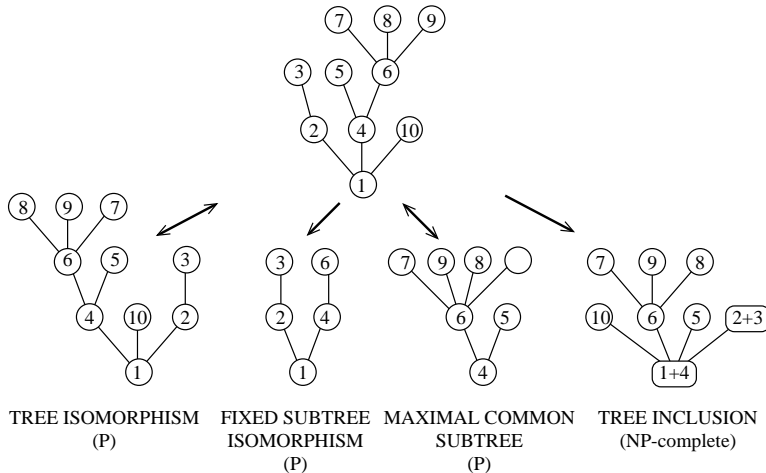


Figure 3: Tree matching problems. The labels illustrate possible solutions.

TREE ISOMORPHISM belongs to **P**. As with isomorphisms in general, **TREE ISOMORPHISM** can be detected by enumeration. Read (1972) reviews ranking algorithms for free trees, rooted trees, and planar (ordered) rooted trees. The basic idea is to sort subtrees recursively, starting from leaves.

Let us define **FIXED SUBTREE ISOMORPHISM** as **SUBGRAPH ISOMORPHISM** between two trees with the additional constraint that *the roots are mapped to each other*. For that problem, Reyner (1977) published a polynomial time algorithm which was based on the outline given by Matula (1968) and later corrected by Verma and Reyner (1989). We prefer using here the word “fixed” instead of “rooted” because the latter ambiguously refers to both matching of rooted trees and root-fixed matching. Pelillo et al. (1999a) have studied subtree isomorphisms, but aiming at finding a **MAXIMAL COMMON SUBTREE** of two trees, that is, the largest connected graph that is contained in both source trees. The subtree does not have to include both roots. The fact that the trees are rooted means only that the directions of the parent-child relations are preserved. Clearly, **MAXIMAL COMMON SUBTREE** is not a subproblem of **FIXED SUBTREE ISOMORPHISM**, nor vice versa, but it stays in **P** as it can be solved with an exact algorithm similar to that of Reyner (1977) for **FIXED SUBTREE ISOMORPHISM**: one may easily replace checking an isomorphism to calculating a maximal sub-isomorphism, and then, applying that through all the $|V_1| + |V_2| - 1$ possible rooting positions.

TREE INCLUSION corresponds to **GRAPH CONTRACTABILITY** but has the additional constraint of ancestorship preservation. Kilpeläinen and Mannila (1995) presented a proof of its **NP-completeness** by a reduction from the fundamental combinatorial problem **SATISFIABILITY**.

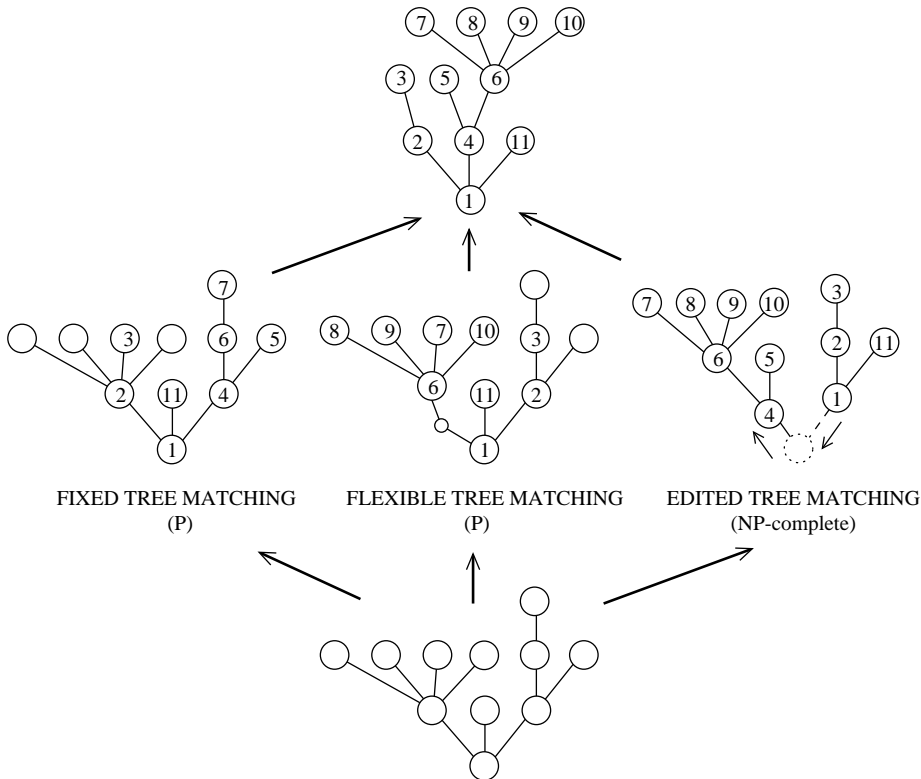


Figure 4: Tree matching problems (continued).

Let us proceed to the three problems illustrated in Fig. 4. These problems have a central role in this thesis. Define *optimal depth-fixed one-to-one rooted unordered tree matching*, or briefly **FIXED TREE MATCHING**, as follows.

Definition 7 (**FIXED TREE MATCHING**). Given trees $T_u = (U, E_u)$ and $T_v = (V, E_v)$, construct tree $T_m = (M, M', E_m) = \text{fixedTreeMatching}(T_u, T_v)$, that has two sets of vertices M and M' , under the following constraints. In addition to being the vertex sets of T_m , M and M' are still, in the spirit of (3) and (4), the matched vertex pairs and unmatched vertices of T_u and T_v as follows:

1. Each vertex $m \in M$ is a pair of matched vertices, $m = (u, v)$.
2. Each $m' \in M'$ is a single vertex, $m' = u$ or $m' = v$.
3. The roots are always matched: $r_m = (r_u, r_v)$.
4. Each u and v is in either M or M' : for each u exists either $m' \in M' : m' = u$ or $(m, v) \in M : m = (u, v)$ for some v ; respectively for v .
5. Parent-child relations stated by E_u and E_v are preserved in E_m : if $m = (u, v)$ then $\text{parent}(m) = (\text{parent}(u), \text{parent}(v))$, or u, v , and m are roots, and if $m' = u$ then

$\text{parent}(m') = \text{parent}(u)$ or $\text{parent}(m') = (\text{parent}(u), v)$ for some v ; respectively for v . (Since the parent relation is unique, this constraint guarantees that T_m is a tree.)

6. The number of matched vertices is maximized. In the case of weighted trees, attribute trees, and weighted attribute trees, also a cost function as defined by (5) and Table 2 is minimized. Inversely, plainly topological matching corresponds to the case where $c(m) \equiv 0$ and $c(m') \equiv 1$.

The above definition is a basis for modifications proposed in the publications of this thesis. Trees can be addressed by their roots, especially in programming. Thus, in the spirit of polymorphism, the notation $(m, d) = \text{fixedTreeMatching}(u, v)$, outputting the root m of T_m and tree distance $d = \text{dist}(T_u, T_v)$, will be used in parallel with the form $T_m = \text{fixedTreeMatching}(T_u, T_v)$ in this thesis.

FIXED TREE MATCHING resembles MAXIMUM COMMON SUBTREE and FIXED SUBTREE ISOMORPHISM as it implies finding a maximum fixed subtree. An algorithm for FIXED TREE MATCHING, outlined by Kilpeläinen and Mannila (1994) and Mannila (2000), based on that of Reyner (1977) for FIXED SUBTREE ISOMORPHISM, is presented on p. 22 (Algorithm 1). First, m is associated with u and v , the roots of the trees to be matched. If neither u nor v has children, no further matching is needed. If only one of u and v has children, the subtrees rooted by the children become embedded as such. If both u and v have children, the tree matching problem is recursively divided into WEIGHTED BIPARTITE MATCHING problems.

The subroutine `weightedBipartiteMatching` is assumed to accept sets of different size, hence the resulting cost matrix \mathbf{D} may be non-square (even empty) and the costs of unmatching, \mathbf{d}' , have to be considered as well. Using the standard formulation of WEIGHTED BIPARTITE MATCHING (p. 16), each vertex pair (u_i, v_j) has cost $\mathbf{D}(i, j)$, and if $|U^+| > |V^+|$ for each vertex $u_i \in U^+$ there is a unique “mismatch vertex” $u'_i \in V^+$ with (u_i, v_j) having cost $\mathbf{d}'(i)$; symmetrically for $|V^+| > |U^+|$. Further, for input $v = \lambda$ (null vertex), a copy of $T(u)$ is attached as such to $T(m)$; symmetrically for $u = \lambda$. For notational convenience, $c(u, \lambda) \equiv c'(u)$ and $c(\lambda, v) \equiv c'(v)$. A bracketed indexing of a vector or a matrix refers to a vector consisting of the elements indexed; `sum()` is the sum of the elements of a vector.

The complexity of FIXED TREE MATCHING is of special interest here. Algorithm 1 is principally the same as Reyner’s (1977) algorithm for SUBTREE ISOMORPHISM but now distances instead of isomorphisms are calculated; while FIXED SUBTREE ISOMORPHISM contains BIPARTITE MATCHING as a subproblem, FIXED TREE MATCHING contains analogously WEIGHTED BIPARTITE MATCHING. Reyner concludes that his algorithm has complexity $\mathcal{O}(N^p)$, where $N = \max(|U|, |V|)$ and p equals the exponent in the complexity $\mathcal{O}(n^p)$ for general BIPARTITE MATCHING of two sets of n vertices. In other words, the overall complexity of FIXED TREE ISOMORPHISM depends on the complexity of the best algorithm found for BIPARTITE MATCHING. Due to analogy, Algorithm 1 for FIXED TREE

Algorithm 1: An exact algorithm for FIXED TREE MATCHING.

$(m, d) = \text{fixedTreeMatching}(u, v)$		
input:	u, v	roots of trees, either of which can be λ (null vertex)
output:	m	root pointing to the matched trees
	d	distance between the trees
	M^+	children of m (implicit; accessible via m)
variables:	U^+, V^+	children of u, v
	\mathbf{M}	matrix of the roots of (potentially) matched trees
	\mathbf{m}'	vector of (potentially) unmatched vertices
	c	vertex cost function
	\mathbf{D}	distance matrix
	\mathbf{d}'	vector of costs of unmatching
	M	set of index pairs of \mathbf{D} referring to matched vertices
	M'	set of indices of \mathbf{d}' referring to unmatched vertices

1. set $m = (u, v)$,
set $M^+ = \emptyset$,
set $d = c(u, v)$;
2. if $U^+ = \emptyset$ and $V^+ = \emptyset$,
return (m, d) ;
3. for each $u_i \in U^+$,
for each $v_j \in V^+$,
 $(\mathbf{M}(i, j), \mathbf{D}(i, j)) = \text{fixedTreeMatching}(u_i, v_j)$;
4. if $|U^+| > |V^+|$,
for each $u_i \in U^+$,
 set $(\mathbf{m}'(i), \mathbf{d}'(i)) = \text{fixedTreeMatching}(u_i, \lambda)$;
else if $|V^+| > |U^+|$,
for each $v_i \in V^+$,
 set $(\mathbf{m}'(i), \mathbf{d}'(i)) = \text{fixedTreeMatching}(\lambda, v_i)$;
5. set $(M, M') = \text{weightedBipartiteMatching}(\mathbf{D}, \mathbf{d}')$;
6. set $M^+ = \mathbf{M}[M] \cup \mathbf{m}'[M']$,
set $d = d + \text{sum}(\mathbf{D}[M]) + \text{sum}(\mathbf{d}'[M'])$;
7. return (m, d) .

MATCHING can be expected to require $\mathcal{O}(N^p)$ operations, where p equals now the exponent of the $\mathcal{O}(n^p)$ complexity for WEIGHTED BIPARTITE MATCHING of two sets having n vertices. As mentioned on p. 17, $p = 3$ in general case.

Intuitively, it seems that the complexity of FIXED TREE MATCHING must be at least quadratic due to the construction of the distance matrices. However, an open question seems to be whether trees possess a property that allows observing only a subset of the distances. The central problems and complexities discussed are collected to Table 3.

The constraint of fixed depths in FIXED TREE MATCHING may be critical in some applications. FLEXIBLE TREE MATCHING is defined similarly to FIXED TREE MATCHING, but now *edges* in both trees can be split by adding intermediate vertices. Intuitively, this means that edges can be “stretched”.

Table 3: Computational complexities.

problem	complexity	reference
BIPARTITE MATCHING	$\mathcal{O}(n^{\frac{3}{2}})$	Hopcroft and Karp (1973)
FIXED SUBTREE ISOMORPHISM	$\mathcal{O}(N^{\frac{3}{2}})$	Reyner (1977)
WEIGHTED BIPARTITE MATCHING	$\mathcal{O}(n^3)$	Lawler (1976)
— planar Euclidean distances	$\mathcal{O}(n^{\frac{3}{2}} \log n)$	Vaidya (1989)
FIXED TREE MATCHING	$\mathcal{O}(N^3)$	by analogy to Reyner (1977)
— heuristic algorithm	$\mathcal{O}(N \log N)$	Section 4

While FLEXIBLE TREE MATCHING allows *edge splitting*, yet more combinatorial freedom — and complexity — is obtained by allowing *vertex splitting*. Shasha et al. (1994) define TREE EDIT DISTANCE as the problem of finding a minimal set of edit operations in transforming a tree T_u to another tree T_v . The operations — **insert**, **delete**, and **change** (Fig. 5) — are generalizations of those used for calculating a *Levenshtein distance* in string matching (Levenshtein 1965, Kohonen 1985). The deletion of a vertex v means that v is removed from V and the children of v become children of $\text{parent}(v)$, the parent of v . An insertion is the inverse of a deletion. Changing a label is obtained by a deletion followed by an insertion, but **change** is typically considered as an operation of its own. Each operation has a cost — conventionally one — and the distance is the minimum total cost. The **NP**-completeness of TREE EDIT DISTANCE was shown by Zhang et al. (1992) by a reduction from EXACT COVER BY 3-SETS (Garey and Johnson 1979). Calculating the distance implies finding an *optimal one-to-one edited rooted unordered tree matching*. Due to this equivalence and for consistency in vocabulary, we prefer calling this problem EDITED TREE MATCHING instead of a TREE EDIT DISTANCE.

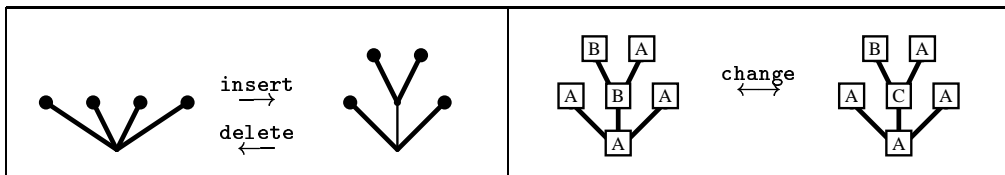


Figure 5: Tree edit operations.

At this point, let us return to the example of Fig. 4 and consider the distances corresponding the matchings. Clearly, in FIXED TREE MATCHING, six vertices remain unmatched: three in the “source” tree (unlabelled vertices) and three in the “target” tree (8,9,10). FLEXIBLE TREE MATCHING catches the crown of four vertices (7,8,9,10), with a cost inserting one vertex (small circle), and leaving still three vertices unmatched (5 and two unlabelled), so the distance is four. Only two operations are needed in EDITED TREE MATCHING for transforming one tree to the other.

Assuming that an application has been given, the question is: among the three matching problems of Fig. 4, which one yields the most natural distance function? Obviously, there is no universal answer. The choice of a matching problem (model), and thereby of

a distance function, depends on the application. However, in the example of Fig. 4 one could intuitively expect that the crowns of four leafs should correspond to each other. In that sense, FIXED TREE MATCHING is incapable of detecting similarity of the trees. On the other hand, the way how EDITED TREE MATCHING finds the correspondence between the trees is somewhat unnatural; the combination of a deletion and an insertion, causing two subtrees to be swapped, looks too powerful.

The emphasis of this thesis is above all on FIXED TREE MATCHING, keeping in mind its relation to the better known, **NP**-complete EDITED TREE MATCHING. FLEXIBLE TREE MATCHING, which in a way falls between the two abovementioned, is not discussed here but will be subject to future studies.

2.5 Syntactic tree recognition

Trees can be also recognized *syntactically*, as proposed by Fu and Bhargava (1973). In computer vision, syntactical approaches are feasible if the studied objects have indeed been generated by a syntactic phenomenon: many man-made objects and some natural objects, like plants, certainly have. Inversely, regularities of botanical trees have also inspired constructing artificial trees (Imiya et al. 1996). One may of course argue that any tree, stored as a list of parent-child relations, *is* a syntax. Essentially, the question is whether a tree contains repeated, self-similar structures (fractals) that could be described compactly by a grammar. The assumption in this thesis is that the objects of interest are such that the size of an extracted tree reflects an inherent property of the modelled object. Nevertheless, it seems that the approach of processing trees as such — by their explicit structure instead of a grammar — is rather general by being capable of handling also trees having syntactic nature, especially those in which the syntax varies as the function of depth in a tree.

3 Heuristic approaches

3.1 General

According to Pearl (1984), *heuristics* are “...popularly known as rules of thumb, educated guesses, intuitive judgments or simply common sense. In more precise terms, heuristics stand for strategies using readily available though loosely applicable information to control problem-solving processes in human beings and machine.”

Pearl’s definition makes no assumption of the exactness of the result of a computation. For computing a problem we may have two exact procedures one of which is faster on average due to prior knowledge or assumptions on encountered problem instances.

Sometimes a heuristic means the opposite of an *algorithm*: an algorithm produces an exact solution with a minor concern on the computation time while a heuristic reduces the computation time and produces a suboptimal result as a trade-off. In the field of computer science, one of the typical tasks is to determine whether a problem is in **P** or **NP**. The problems and the inputs of the problems are assumed to be well-defined. Any polynomial algorithms, with less concern on their degree, are generally considered practically “good” or “tractable”. In the field of pattern recognition, the goal is more in the direction of finding feasible solutions for natural — incomplete, fuzzy or approximate — data. In this thesis, “algorithm” loosely refers to any computational procedure, heuristic or not. Most of the problems addressed are optimization problems rather than “yes-no” decision problems.

There are two types of heuristics related to this research. *Randomized algorithms* (Motwani and Raghavan 1997) which essentially replace exhaustive trials of combinations by testing subsets of combinations. In *approximation algorithms* (Papadimitriou 1994) the emphasis is in approximability, that is, in controlling the upper bound of error. Given an **NP**-complete problem, heuristic algorithms of polynomial complexity are considered. For example, **NODE COVER** is an **NP**-complete problem in which one tries to find a minimal subset V' of vertices (nodes) in a graph $G = (V, E)$ such that each edge $e \in E$ has at least one endpoint in V' ; the *best currently known* polynomial approximation algorithm finds a subset that has maximally twice the cardinality of the correct V' for any graphs (Papadimitriou 1994). The simplicity of the algorithm is surprising: edges (vertex pairs) are deleted in random order until no edges remain. Generally, finding an error bound for a solution or even deciding whether such a bound exists is often a challenging problem (see Kann 1992).

3.2 Heuristics for matching problems

Before proceeding to the proposed heuristic tree matching scheme in Sec. 4, recent heuristic approaches in matching problems are reviewed in the following.

As mentioned in Sec. 2.3, ranking is one of the exact techniques for detecting isomorphisms. However, ranking is unable to assess similarity of objects in continuous terms. A heuristic counterpart for ranking is to extract an easily computable characteristics for each object. The characteristics are not assumed to be unique, but nevertheless distinguishing enough for a specific application. Such characteristics are often called *indices* as in this thesis or *fingerprints* (Motwani and Raghavan 1997) or *invariants* (Kreher and Stinson 1999). A *feature* is a standard term in pattern recognition but not used here in order to avoid confusion with tree attributes. Unlike with ranks, several different targets may correspond to the same index. Indices may fail in detecting isomorphisms but perform well in more continuous similarity assessments as indices can be designed to carry topological information.

The challenge is to design simple but informative indices. Herbin (1995) has proposed using a neural network, the Self-Organizing Map (Kohonen 1990) for solving BIPARTITE MATCHING. The basic idea is to use weighted sum of topological adjacencies as indices of a graph. A similar approach has been used by Choy and Siu (1995) for a related decision problem, BIPARTITE SUBGRAPH PROBLEM (Garey and Johnson 1979).

The proposed heuristic, discussed in the next section, computes and applies tree indices recursively. Tree indices have been discussed also by Shokoufandeh et al. (1999) in the context of tree matching based database queries. However, the indices are based on eigenvalues of adjacency matrices and are hence less explicit than the simple indices proposed in this thesis. In addition, the indices are computed for the root only; it is unclear whether the indices of Shokoufandeh et al. could be computed recursively, in order to use them in heuristic matching. Shokoufandeh et al. recommended using the indices for limiting the input set for the actual, independent tree matching process.

Generally, the “indices” do not have to be numbers or sets of numbers. Eshera and Fu (1984) suggested simple subgraphs as characteristics of directed attribute graphs; the overall degree of similarity was calculated as the number of operations in recomposing the objects from the subgraphs.

One of the standard techniques in solving a problem in **NP** is to reduce it to another problem in **NP** for which heuristics have been already developed. For example, the famous TRAVELLING SALESPERSON problem is **NP**-complete (Garey and Johnson 1979) and has been a benchmark challenge for designers of heuristics. Although TRAVELLING SALESPERSON is not an actual matching problem, it is a problem dominated by topology. Hence, it can be approximated using the topological nature of the Self-Organizing Map (Fujimura et al. 1999). In principle, one could also try reducing a matching problem to an instance of TRAVELLING SALESPERSON.

Pelillo et al. (1999b,2000) suggest a similar approach for MAXIMAL COMMON SUBTREE. There is also an interesting extension for attribute trees (Pelillo et al. 1999a). The reduction used by Pelillo et al. is based on the fact that GRAPH ISOMORPHISM can be solved by transforming the original graph to a so called *association graph* and then solving an **NP**-complete problem MAXIMUM CLIQUE for that graph. For MAXIMAL COMMON SUBTREE, an additional constraint is applied in the reduction for preserving the hierarchy of trees. For solving the intermediate problem, MAXIMUM CLIQUE, a continuous optimization scheme is applied.

In principle, expanding a problem from **P** to **NP** makes little sense, but Pelillo et al. motivate the approach by noting that a new instance preserves the simplicity of the original problem, and further assume that the related tools — essentially designed to attack harder (**NP**-complete) problems — will adapt to the simplicity. This property is not proven formally but the experiments suggest that “the basins of attraction tend to be large”, that is, the optimization process in the continuous space is mostly stable and converges to an optimal or nearly optimal result. Theoretically, though **NP**-complete problems are reducible to each other, producing equivalent formulations of a *decision problem*, the reductions do not generally transfer the approximabilities of the corresponding *function problems* (Kann 1992).

Nevertheless, the approach of Pelillo et al. can be criticized because of the fact that it involves an expansion of the input trees $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ to a graph $G = (V, E)$ having $V = V_1 \times V_2$, and then calculations using the adjacency matrix, thus a matrix of size $|V|^2 = |V_1 \times V_2| \times |V_1 \times V_2|$. Hence the computational complexity is of $\mathcal{O}(N^4)$, denoting $N = \max(|V_1|, |V_2|)$.

Along proposing an exact algorithm for EDITED TREE MATCHING, Shasha et al. (1994) proposed using three heuristic techniques: Iterative Improvement, Simulated Annealing, and Two Phase Heuristic. All the heuristics treat the problems as a state space. Iterative Improvement simply picks a random state and then continues to such randomly chosen neighbouring states which imply a decrement in a cost function. Simulated Annealing is similar but avoids getting stuck to local minima by proceeding, with some small probability, also to the directions of increasing cost. Two Phase Heuristic is a mixture of the first two. Using these three methods for trees having from 10 to 20 vertices, Shasha et al. obtained distance approximations with 9%...18% error. Slightly better results were obtained by sorting the subtrees rooted to each vertex by their descendant count, and using the resulting order as an initial state.

An alternative for solving exact problems heuristically is to define the problems less strictly in the first place. Some tree-related approaches used in computer vision are discussed in Sec. 7.

4 Proposed approach: index-based matching

This section introduces briefly a heuristic procedure for `FIXED TREE MATCHING`. The procedure, including slight variations, is also explained in the publications of this thesis but is reviewed here for consistency. Likewise, only weighted attribute trees will be considered. The weight of vertex v will be denoted as $w(v)$, or briefly w_v and attributes (attribute vectors) as $\mathbf{a}(v)$ or \mathbf{a}_v . The weight and the attribute (vector) of vertex v_i will be denoted w_i and \mathbf{a}_i , respectively. On the other hand, an unweighted, unattributed tree can be obtained by setting $w \equiv 1$ and $\mathbf{a} \equiv 1$.

4.1 Basic idea

The exact procedure for `FIXED TREE MATCHING` presented in Algorithm 1 is an exhaustive depth-first computation: a match between two vertices can be established or rejected only after the respective children are completed, recursively.

In the proposed heuristic procedure, Algorithm 2, the basic idea is to avoid exhaustive calculations by *indexing*. Before actually matching two trees, each vertex is attached an index vector that describes the subtree rooted at that vertex. In the matching stage, subtrees become matched according to their indices. The matching starts from the roots and proceeds recursively upwards in one pass; no backtracking is applied. One step of the procedure is shown in Fig. 6.

As with Algorithm 1, it is convenient to present the detected correspondences between trees T_u and T_v by tree T_m , the vertices of which are the matched and unmatched vertices of T_u and T_v . Since in each vertex v_m the recursion proceeds *after* establishing the match, the complexity of the core algorithm is $\mathcal{O}(N)$, where $N = \max(|U|, |V|)$. The overall complexity of matching, discussed at the end of Sec. 4.3, depends however on the complexities of indexing and `WEIGHTED BIPARTITE MATCHING` of subtree indices.

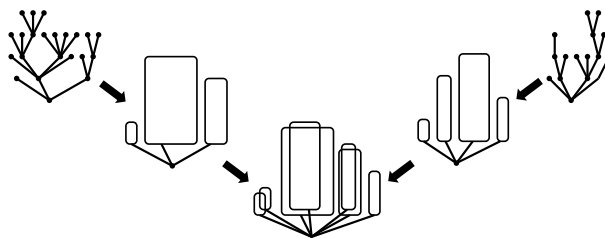


Figure 6: Index-based matching. The subtrees are matched relying on the information carried by indices. In this example, two indices — height and vertex count — are applied and illustrated as rectangles having the respective heights and widths.

Algorithm 2: A heuristic algorithm for FIXED TREE MATCHING. The basic difference to Algorithm 1 is that bipartite matching is performed on approximated $\hat{\mathbf{D}}$, and `fixedTreeMatching` is called *after* `weightedBipartiteMatching`.

$(m, d) = \text{fixedTreeMatching}(u, v)$		
input:	u, v	roots of trees
output:	m	root pointing to the matched trees
	d	distance between the trees
	M^+	children of m (implicit; accessible via m)
variables:	U^+, V^+	children of u, v
	c	vertex cost function
	\hat{c}	approximated, unscaled tree-to-tree distance (index distance)
	$\hat{\mathbf{D}}$	approximated, unscaled distance matrix (index distance matrix)
	$\hat{\mathbf{d}}'$	vector of approximated costs of unmatching
	M	set of index pairs of $\hat{\mathbf{D}}$ referring to matched vertices
	M'	set of indices of $\hat{\mathbf{d}}'$ referring to unmatched vertices

1. set $m = (u, v)$,
 set $M^+ = \emptyset$,
 set $d = c(u, v)$;
2. if $U^+ = \emptyset$ and $V^+ = \emptyset$,
 return (m, d) ;
3. for each $u_i \in U^+$,
 for each $v_j \in V^+$,
 set $\hat{\mathbf{D}}(i, j) = \hat{c}(u_i, v_j)$;
4. if $|U^+| > |V^+|$,
 for each $u_i \in U^+$,
 set $\hat{\mathbf{d}}'(i) = \hat{c}'(u_i)$;
 (respectively for V^+)
5. set $(M, M') = \text{weightedBipartiteMatching}(\hat{\mathbf{D}}, \hat{\mathbf{d}}')$;
6. for each $(i, j)_k \in M$,
 set $(m_k, d_k) = \text{fixedTreeMatching}(u_i, v_j)$,
 set $M^+ = M^+ \cup m_k$,
 set $d = d + d_k$;
7. if $|U^+| > |V^+|$,
 for each $i' \in M'$,
 set $(m_{i'}, d_{i'}) = \text{fixedTreeMatching}(u_{i'}, \lambda)$,
 set $M^+ = M^+ \cup m_{i'}$,
 set $d = d + d_{i'}$;
 (respectively for V^+)
8. return (m, d) .

4.2 Indices

An index is a numerical description of a tree, for instance the height or vertex count of a tree. The topological indices used in this study are collected to Table 4, where the notations are slightly revised from those of the publications for overall consistency.

Table 4: Topological indices. Generally, an index $I(v)$ describes the whole tree rooted at v . The weighted counterpart of $I(v)$ is denoted by $\tilde{I}(v)$. In this table, only the arguments different from v are explicitly shown. The children and the descendants of v are denoted by v_i and v_i^* , respectively. Vertex v and its descendants are collectively denoted by v^{**} .

		Basic definition	Inductive definition
Level	L	$ \text{path}(r, v) $	$L[\text{parent}(v)] + 1$
	\tilde{L}	$\sum_{v' \in \text{path}(r, v)} w(v')$	$\tilde{L}[\text{parent}(v)] + w$
Height	H	$\max[L(v_i^*)] - L$	$\max[H(v_i)] + 1$
	\tilde{H}	$\max[\tilde{L}(v_i^*)] - \tilde{L}$	$\max[\tilde{H}(v_i)] + w$
Child count	D	$\#\{v_i\}$	
	\tilde{D}	$\sum w(v_i)$	
Descendant count	S	$\#\{v_i^*\}$	$D + \sum S(v_i)$
	\tilde{S}	$\sum w(v_i^*)$	$\tilde{D} + \sum \tilde{S}(v_i)$
Vertex count	N	$S + 1$	
	\tilde{N}	$\tilde{S} + w$	
Sum of squared child count	S_2	$D^2 + \sum D(v_i^*)^2$	$D^2 + \sum S_2(v_i)$
	\tilde{S}_2	$\tilde{D}^2 + \sum \tilde{D}(v_i^*)^2$	$\tilde{D}^2 + \sum \tilde{S}_2(v_i)$
Variance of child count	σ^2	$\frac{1}{N} \sum [D(v_i^{**}) - \frac{1}{N} \sum D(v_i^{**})]^2$	$\frac{S_2}{N} - (\frac{S}{N})^2$
	$\tilde{\sigma}^2$	$\frac{1}{N} \sum [\tilde{D}(v_i^{**}) - \frac{1}{N} \sum \tilde{D}(v_i^{**})]^2$	$\frac{\tilde{S}_2}{N} - (\frac{\tilde{S}}{N})^2$
Std.dev. of child count	σ	$\sqrt{\sigma^2}$	
	$\tilde{\sigma}$	$\sqrt{\tilde{\sigma}^2}$	
Torque	T	$\sum [L(v_i^*) - L]$	$S + \sum T(v_i)$
	\tilde{T}	$\sum [L(v_i^*) - L] \times w(v_i^*)$	$\tilde{S} + \sum \tilde{T}(v_i)$
Centroid	C	T/S	
	\tilde{C}	\tilde{T}/\tilde{S}	

The *level* of a vertex, the distance from the root, is as such a noninformative index in FIXED TREE MATCHING but it simplifies the definitions of other indices. The *height* is simply the distance from the root to the highest vertex. The *child count* is the number of vertices adjacent to the root; the *descendant count* is the respective cumulative sum. It is practical to have a separate notation for the *vertex count*, even though the only difference is the contribution of the root. The *squared descendant count* is perhaps an ambiguous yet convenient short of *the sum of squared child counts*, and it is used in calculating the *variance of child count* and the *standard deviation of child count*, which are measures of the irregularity of a tree topology. The *torque* and *centroid* take large values for trees in which relatively many vertices are by the top of the tree.

As a tree can be referred to by its root, indices of the tree can be likewise associated with the root. Thus, assuming $T = T(v)$, without confusion one may denote an index by $I(v)$ instead of $I(T)$. Several indices can be collected as an index vector $\mathbf{i} = [I_1 \ I_2 \ \dots]$.

In order to keep the overall computation time of heuristic matching tractable, the indexing phase should be simple. The scheme presented here is linear, $\mathcal{O}(N)$, thus at least not exceeding the complexity of the core algorithm. The key idea is to use cumulative indices, or more generally, indices that can be derived *inductively*, that is, an index of a vertex is a function of the indices of its children.

An inductively computable index can be a function of any indices of the children. The derivation of indices listed in Table 4 is presented schematically in Fig. 7. For example, the weighted standard deviation of child count, $\tilde{\sigma}$, is obtained from the weighted variance, which is obtained from the weighted descendant count \tilde{S} , squared weighted descendant count \tilde{S}_2 , and total vertex count N . The indices connected by a solid line are linearly dependent. For example, the weighted torque \tilde{T} of a tree is a linear sum of the weighted descendant counts and the torques of the children of the root. On the other hand, the dotted line from \tilde{S} and centroid \tilde{C} tells that the latter is nonlinear. Linearity of indices is a desired property of weighted tree mixtures discussed in Sec. 5.3.

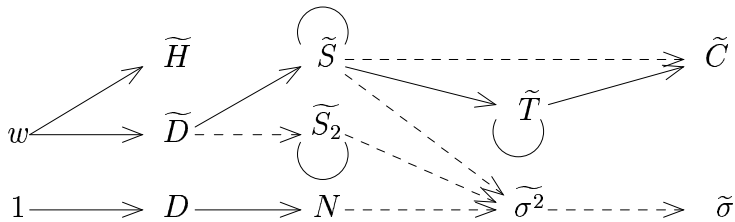


Figure 7: Inductive derivation of indices.

In matching attribute trees, indices are needed for describing the distribution of the attributes, too. Some basic *attribute indices* are suggested in Table 5. In some applications, the objects having relevance with respect to recognition may reside (only) at the leaves of a tree. Then, attribute indices can be designed correspondingly, propagating information from leaves only.

Table 5: Attribute indices. An attribute of a vertex is denoted by a ; the attributes of the corresponding descendants and children are denoted by a_i^* and a_i , respectively.

Attribute a	Basic definition	Inductive definition
Maximum A_{\max}	$\max(a_i^*)$	$\max[a_i, A_{\max}(a_i)]$
Minimum A_{\min}	$\min(a_i^*)$	$\min[a_i, A_{\min}(a_i)]$
Mass A_m	$a + \sum a_i^*$	$a + \sum A_m(a_i)$
	\tilde{A}_m	$wa + \sum \tilde{A}_m(a_i)$
Average A_{avg}	$\frac{1}{N} A_m$	
	\tilde{A}_{avg}	$\frac{1}{N} \tilde{A}_m$

4.3 Weighted bipartite matching of indices

Like the exact algorithm for `FIXED TREE MATCHING` (Alg. 1), also the heuristic solution (Alg. 2) contains `WEIGHTED BIPARTITE MATCHING` (p. 16) as a subproblem, the computational complexity of which is $\mathcal{O}(N^3)$. It should be noted that now the input — the indices — are vectors, that is, points in space. If two indices are used, the space is planar and the subproblem can be exactly solved in the reasonable $\mathcal{O}(n^{\frac{5}{2}} \log n)$ time (Table 3).

However, it is natural to apply a heuristic approach also in `WEIGHTED BIPARTITE MATCHING`. Since the items to be matched — the indices — are themselves approximations, it makes little sense to insist on full accuracy in matching them. Instead, one aims at designing efficient heuristics that will fit the simplicity of the core algorithm.

A natural choice for index-to-index distance function is the L -norm (Def. 2)

$$\hat{c}(u, v) = \|\mathbf{i}(u) - \mathbf{i}(v)\|. \quad (7)$$

Consequently, an analogous function for an unmatched vertex (5) is the cost by magnitude,

$$\hat{c}'(u) = \|\mathbf{i}(u)\|. \quad (8)$$

Although the distances between indices are desired to correlate with the actual tree-to-tree distances, it should be noted that the index distance functions do not have to resemble the tree-to-tree distance functions (see (5) and Table 2). For example, one could use fixed cost, say a large positive constant, for unmatched vertices, regarding all the vertices — essentially, subtrees — equivalent in that respect. Alternatively, one could match an unmatched vertex virtually to its nearest (already matched) vertex, and the unmatching cost would equal such virtual matching cost.

In the following, two heuristics for `WEIGHTED BIPARTITE MATCHING` are discussed. Related heuristics have been reviewed by Avis (1983), for example. Here we assume that the indices somehow reflect the size of the tree, say vertex count or height. The vertex sets to be matched are assumed to be U and V , denoting $N = \max(|U|, |V|)$.

Greedy matching. An obvious greedy approach is to match U and V in the ascending order of $c(u, v)$. Basically, the matching problem becomes a sorting problem. Greedy matching tends to start matching from the smallest subtrees, leaving large trees unmatched. Consequently, it seems advisable to favor large trees in matching. In *scaled greedy matching*, the indices are normalized among siblings to the interval $[0, 1]$, guaranteeing that the “smallest” subtree of one tree will be matched to the “smallest” tree of the other tree, and respectively for the “largest” trees. Scaled greedy matching is motivated by some applications involving dynamics, where young objects have similar elements as old objects, but the elements are smaller. Such objects appear for example in the main application related to this research, in monitoring clouds in weather radar images, Sec. 7.

Greedy matching is easy to implement. One calculates the distance array, the size of which is $\mathcal{O}(D^2)$, and picks the D smallest row/column-independent elements in order. For instance, a *mergesort* requires $\mathcal{O}(D^2 \log D)$ operations for sorting D^2 items.

Rank-based matching. The large trees can be prioritized in matching by applying *maximal matching*: vertices are ranked in the order of index vector magnitudes, and then items are matched by descending rank. Similar strategy, with descendant counts, was used also by Shasha et al. (1994) in the initialization of EDITED TREE MATCHING. Correspondingly, *minimal matching* means that small trees are matched first. Ranking (sorting) a set of D children can be carried out in $\mathcal{O}(D \log D)$ time. If several indices are applied, the total ranking could be based on, say, an L -norm of index vectors.

4.4 Overall complexity

Finally, we consider the overall complexity. Assume that the larger tree to be matched has N vertices, and the maximal child count is \bar{D} . Indexing takes $\mathcal{O}(N)$ computation steps and the core of Algorithm 2 visits each of the N vertices once, calculating WEIGHTED BIPARTITE MATCHING for the children of each vertex. Hence, with rank-based or exact WEIGHTED BIPARTITE MATCHING of children one seems to arrive at $\mathcal{O}(N\bar{D} \log \bar{D})$ or $\mathcal{O}(N\bar{D}^3)$ complexity, respectively. A more careful analysis, as pointed out by Kilpelainen (2001), yields even stricter bounds. The overall efforts are

$$\sum_{i=1}^N D_i \log D_i \leq \log \bar{D} \sum_{i=1}^N D_i = (N-1) \log \bar{D} = \mathcal{O}(N \log \bar{D}). \quad (9)$$

and

$$\sum_{i=1}^N D_i^3 \leq \bar{D}^2 \sum_{i=1}^N D_i = \bar{D}^2 (N-1) = \mathcal{O}(N\bar{D}^2). \quad (10)$$

Naturally, efficiency of any heuristic should not be judged only by the computational complexity but by the overall performance, taking the suboptimality of approximative matching into account as well. Especially, as tree matching is often used as a preprocessing for further operations (see Fig. 1), and any errors made in matching will propagate throughout the scheme. The success rate of approximative matching is discussed in Sec. 6. Next, we will discuss a domain in which the speed of (9) is especially attractive.

5 Trees as object representations in learning systems

This section explains how trees can be used as learning objects or prototypes in adaptive pattern recognition systems. A learning process means that an object gradually adapts to given samples based on a similarity measure. We shall limit to the schemes in which the learning is carried out by objects that have the same mathematical representation as the input data, although other schemes of learning do exist. Likewise, we will ultimately focus on continuous rather than discrete learning. The central issue here is how the continuity of trees is implemented, and what is the relation of continuity to matching, learning and distance functions. Specifically, some tree-matching-based merging operations applicable in learning processes are presented. It is also useful to know the limits of the applied operations, especially the extent and accuracy to which the operators obey the intuitive ideas of adaptation. Thus, the consistency and related properties of matching are discussed. Finally, a learning system, the Self-Organizing Map of Attribute Trees, is presented as an demonstration of the proposed learning techniques.

5.1 The role of weights

One of the basic techniques presented in this thesis is to apply trees in which the vertices are weighted. This far, it has been merely assumed that the concerned trees *may* have vertex weights. It is clear that the interpretation of a weight depends on the application. Especially in the context of learning systems, we may expect the following intuitive properties.

1. The weight w of a vertex is an indication of the significance of the vertex. Depending on the application, a weight may be also called the value, probability, confidence or degree of existence of the vertex.
2. Typically $w \in [0, 1]$, and the default value of a weight is 1. A simple (unweighted) tree can be regarded as a weighted tree with the weights equal to unity. Correspondingly, a vertex with zero weight can be identified with a nonexistent vertex. Sometimes $w \in [0, \infty]$ might be applicable. Then, $w = n$ can for instance indicate that the vertex is equal to n vertices each having $w = 1$.
3. In learning systems designed for tree data, the training effect of a sample tree with large weights is stronger than that of a tree with small weights.
4. In evaluating distance functions as minimized total costs as in (6), it is natural to associate large costs to unmatched or poorly matched heavy vertices. On the other hand, in some applications the vertex matching costs could be proportional (also) to the difference of the vertex weights (see Table 2).

5.2 Merging

Typically, learning schemes involve a limited set of adaptive objects, *prototypes*, and the set is often called a *codebook*. The codebook is *trained* by presenting samples repeatedly to the prototypes. The essential goal is to generalize data, that is, to use each prototype for representing several training samples. Generalization is a means for keeping the codebook limited in size.

Practically, a learning system involving trees as adaptive objects requires a procedure for *merging* two trees. In the following, we will present some alternative methods. The learning operations for trees discussed in the following assume that a tree matching algorithm is given, and unless stated otherwise, the applied tree matching algorithm may be exact or heuristic.

Consider the tree obtained by matching two trees: $T_m = \text{fixedTreeMatching}(T_1, T_2)$. (In this section, the form $T_m = \text{fixedTreeMatching}(T_1, T_2)$ is more convenient than the respective “programming notation” $(m, d) = \text{fixedTreeMatching}(u, v)$ applied in Algorithms 1 and 2.)

It should be noted that as an object, T_m is not a completed mixture of T_1 and T_2 but more like a *relation* listing the correspondences between T_1 and T_2 . Hence, we need merging operations that proceed from T_m towards new objects which are of similar kind with T_1 and T_2 . To start with, one can consider a *union* and an *intersection* of two trees as basic merging operations. The definitions presented below assume implicitly that $T_m = (M, M', E_m) = \text{fixedTreeMatching}(T_1, T_2)$ is given.

Definition 8 (Union). The *union of two trees* can be defined as $T_1 \cup T_2 =$

$$T = (V, E, w, \mathbf{a}) : \begin{cases} V & = M \cup M' \\ E & = E_m \\ w(v) & = \begin{cases} \max(w_1, w_2), & \text{if } v = (v_1, v_2), \\ w_1, & \text{if } v = v_1, \\ w_2, & \text{if } v = v_2. \end{cases} \\ \mathbf{a}(v) & = \begin{cases} \max(\mathbf{a}_1, \mathbf{a}_2), & \text{if } v = (v_1, v_2), \\ \mathbf{a}_1, & \text{if } v = v_1, \\ \mathbf{a}_2, & \text{if } v = v_2. \end{cases} \end{cases} \quad (11)$$

where $v_1 \in V_1$ and $v_2 \in V_2$, and respective abbreviations $w_i = w(v_i)$ and $\mathbf{a}_i = \mathbf{a}(v_i)$ for $i = 1, 2$ are used for convenience, and $\max(\mathbf{a}_1, \mathbf{a}_2)$ is taken element-wise.

Definition 9 (Intersection). Correspondingly, the *intersection* is $T_1 \cap T_2 =$

$$T = (V, E, w, \mathbf{a}) : \begin{cases} V & = M \\ E & = E_m \cap (M \times M) \\ w(v) & = \min(w_1, w_2) \\ \mathbf{a}(v) & = \min(\mathbf{a}_1, \mathbf{a}_2) \end{cases} \Bigg\} \text{ if } v = (v_1, v_2) \quad (12)$$

Now, we can define the subset relation $T_1 \subseteq T_2 \Leftrightarrow [(T_1 \cap T_2) = T_1] \Leftrightarrow [(T_1 \cup T_2) = T_2]$. For example, a learning system could classify a given sample T' to a class c if T' fits between the set intersection and the set union generated for that class: $\bigcap_c T \subseteq T' \subseteq \bigcup_c T$.

More sophisticated merging is obtained if trees can be *mixed continuously*. In a learning system, a prototype could asymptotically adapt towards some kind of an average of the samples representing a certain class. With vectors, one often applies update rules of the type $\mathbf{m}'_i = \mathbf{m}_i + \alpha(\mathbf{x} - \mathbf{m}_i)$ or equivalently, $\mathbf{m}'_i = \alpha\mathbf{x}_i + (1 - \alpha)\mathbf{m}_i$, where \mathbf{x} is a sample, \mathbf{m}_i is a prototype and α is a *learning coefficient*.

Next, our goal is to define the corresponding rule for trees.

Definition 10 (Mixture of trees). Using the same assumptions and notations as above, the *mixture* of trees T_1 and T_2 with mixing coefficient $\alpha \in [0, 1]$ is defined as

$$\text{mixTrees}(T_1, T_2, \alpha) = \begin{cases} V & = M \cup M' \\ E & = E_m \\ w(v) & = \begin{cases} \alpha w_1 + (1 - \alpha)w_2, & \text{if } v = (v_1, v_2), \\ \alpha w_1, & \text{if } v = v_1, \\ (1 - \alpha)w_2, & \text{if } v = v_2. \end{cases} \\ \mathbf{a}(v) & = \begin{cases} \frac{\alpha w_1}{\alpha w_1 + (1 - \alpha)w_2} \mathbf{a}_1 + \frac{(1 - \alpha)w_2}{\alpha w_1 + (1 - \alpha)w_2} \mathbf{a}_2, & \text{if } v = (v_1, v_2), \\ \mathbf{a}_1, & \text{if } v = v_1, \\ \mathbf{a}_2, & \text{if } v = v_2. \end{cases} \end{cases} \quad (13)$$

Thus, a mixture blends two trees in a continuous fashion: $T_\alpha = \text{mixTrees}(T_1, T_2, \alpha)$ represents T_1 and T_2 in proportions α and $1 - \alpha$, respectively, T_α can be seen as a *weighted instantiation* of the respective matching, $T_m = \text{fixedTreeMatching}(T_1, T_2)$, keeping in mind that T_α is an object analogous to T_1 and T_2 , while T_m has the nature of a relation. Strictly speaking, T_1 and T_2 may be unweighted, but T_α is always weighted.

Intuitively, (13) realizes an *interpolation*. It should be however seen as one among many possible definitions of tree mixtures or interpolations; a detailed discussion on its properties is presented in 5.3. In principle, the learning systems dealing with scalars and vectors could be generalized for trees if one could extend appropriate arithmetical operands for trees. For example, one could think of rewriting the mixture of two trees more generally as

$$\text{mixTrees}(T_1, T_2, \alpha) = \alpha T_1 + (1 - \alpha)T_2. \quad (14)$$

The following definitions of multiplication and addition are steps towards this goal. Obviously, the left-hand side of (14) is continuous with respect to α because mixing takes affect *after* matching while the right-hand side of (14) suggests that weighting is carried out *before* matching, possibly affecting the matching stage: small changes in α may cause recombinations in M , the set of matched vertices, and thus, clear discontinuities. The conditions under which (14) holds are discussed in 5.3.

Definition 11 (Scalar multiplication). Given a weighted attribute tree $T = (V, E, w, \mathbf{a})$ and a scalar α , a *scalar multiplication* is a tree in which all the weights have been multiplied by α :

$$\alpha \cdot T = (V', E', w, \mathbf{a}) : \begin{cases} V' = V, \\ E' = E, \\ w(v') = \alpha \cdot w(v), \\ \mathbf{a}(v') = \mathbf{a}(v). \end{cases} \quad (15)$$

Definition 12 (Summation of trees). Given two weighted attribute trees $T_1 = (V_1, E_1, w_1, \mathbf{a}_1)$ and $T_2 = (V_2, E_2, w_2, \mathbf{a}_2)$, the *addition of two trees* equals matching them and summing the weights and the attributes

$$T_1 + T_2 = (V, E, w, \mathbf{a}) : \begin{cases} V & = M \cup M' \\ E & = E_m \\ w(v) & = \begin{cases} w_1 + w_2, & \text{if } v = (v_1, v_2) \in M, \\ w_1, & \text{if } v = v_1 \in M', \\ w_2, & \text{if } v = v_2 \in M', \end{cases} \\ \mathbf{a}(v) & = \begin{cases} \frac{w_1}{w_1+w_2} \mathbf{a}_1 + \frac{w_2}{w_1+w_2} \mathbf{a}_2, & \text{if } v = (v_1, v_2) \in M, \\ \mathbf{a}_1, & \text{if } v = v_1 \in M', \\ \mathbf{a}_2, & \text{if } v = v_2 \in M'. \end{cases} \end{cases} \quad (16)$$

Remark. We do not require a tree summation to obey familiar properties like associativity. Related topics could, however, be subject to further studies.

5.3 Consistency of matching and merging

Learning systems are compositions of operations and data structures. For overall control, error tracking and sensitivity analysis, it is practical to require that the modules of such system are robust and easy to analyse. For example, it is often hard to optimize a learning system if it involves many stochastic or heuristic steps. In learning systems involving index-based tree matching, one can simplify the overall control for example by concentrating on index design while keeping the procedure of actual matching simple and but robust.

In designing a learning system, it should be kept in mind that the performance of mixing depends on the compatibility between distances ((6), Table 2) and mixing operations (13), but essentially on the performance of matching. Further, if the continuity of mixtures is a desired property, one should know how accurately it applies in practice. In addition to continuity, there are properties like stability and linearity which may be crucial when designing a learning system. In applications involving prediction of trees, also differentiability might be of interest. Some possible shapes of tree interpolation curves are illustrated in Fig. 8. The ultimate question is: to which extent do tree interpolations in tree space correspond to vector interpolations in Euclidean space?

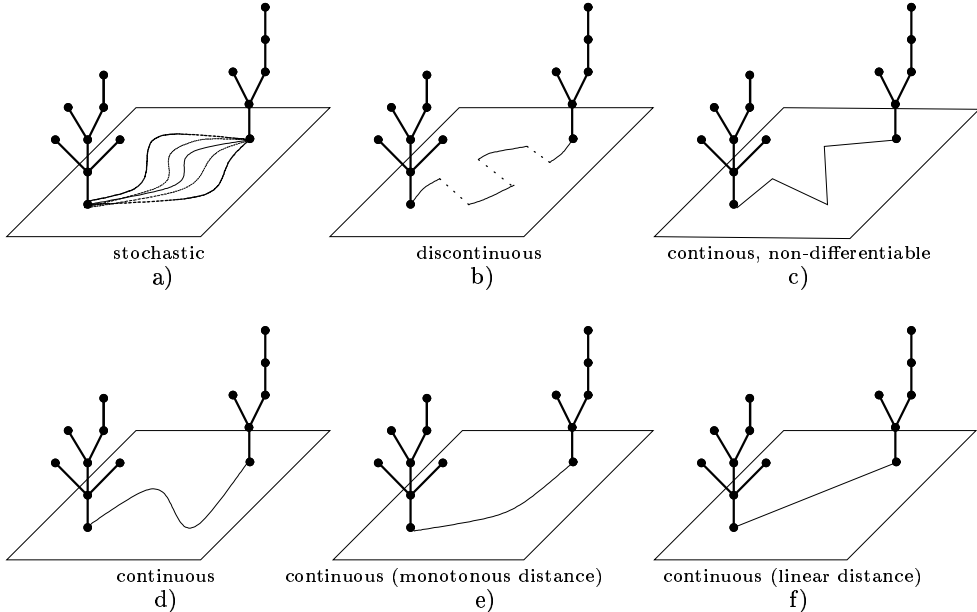


Figure 8: Hypothetical interpolation paths in tree space illustrated as a plane.

The rest of this section will try answer the above question, discussing how matching schemes tolerate prior weighting of trees (consistency of matching), how stable the obtained mixtures are (consistency of mixing) and how well tree mixtures follow the intuitive idea of interpolation (linearity of distances).

Definition 13 (Consistency of matching). A matching scheme is *consistent* if positive scalar multiplication has no effect on matching. Formally, assume $T_m = (M, M', E_m) = \text{fixedTreeMatching}(T_1, T_2)$ and $T_\alpha = (M_\alpha, M'_\alpha, E_\alpha) = \text{fixedTreeMatching}(\alpha_1 T_1, \alpha_2 T_2)$. Then, fixedTreeMatching is consistent if $M = M_\alpha$ for any attribute trees T_1 and T_2 and for any $\alpha_1 > 0$ and $\alpha_2 > 0$.

Consistency of matching is essential for example in applications where one expects that weak trees gradually adapt to topologically similar but stronger trees without recombinations among M ; recombinations may slow down or corrupt the process of adaptation.

Lemma 1 (Linearity of indexing). Given tree T , let $\mathbf{i}(T)$ be a vector composed of (some of) the indices \tilde{D} , \tilde{S} , \tilde{N} , and \tilde{T} listed in Table 4. Then $\mathbf{i}(T)$ is linear, that is,

$$\begin{aligned} \mathbf{i}(\alpha T) &= \alpha \cdot \mathbf{i}(T) \text{ and} \\ \mathbf{i}(T_1 + T_2) &= \mathbf{i}(T_1) + \mathbf{i}(T_2). \end{aligned} \tag{17}$$

Proof. The indices \tilde{D} , \tilde{S} , \tilde{N} , and \tilde{T} are linear combinations of vertex weights $w(v)$, in other words, a sum of the type $\sum_{v_i} b_i w(v_i)$ where b_i are constants. This property can be seen in Table 4, or more visually in Fig. 7. Scaling each w to αw scales also the

linear combination by α . Likewise, according to the definition of summation (Def. 12), the weights in T are simply sums of the weights of matched vertices. However, looking at the definitions of \tilde{D} , \tilde{S} , \tilde{N} , and \tilde{T} in Table 4, there is no difference between handling weights as sums and separately. \square

Remark. Indices \tilde{H} , \tilde{S}_2 , $\tilde{\sigma}^2$, and \tilde{C} are nonlinear. For example, $\tilde{S}_2(\alpha T) = \alpha^2 \tilde{S}_2(T)$, $\tilde{\sigma}^2(\alpha T) = \alpha^2 \tilde{S}_2/N - \alpha^2 (\tilde{S}/N)^2 = \alpha^2 \tilde{\sigma}^2(T)$, and $\tilde{C}(\alpha T) = \tilde{C}(T)$. However, the first criterion applies to \tilde{H} and $\tilde{\sigma}$. For example, $\tilde{\sigma}(\alpha T) = \sqrt{\tilde{\sigma}^2(\alpha T)} = \alpha \tilde{\sigma}(T)$.

The following lemma means that we can restrict our scope to the consistency of children-to-children matching:

Lemma 2 (Universality of consistency). If an index-based matching scheme applying linear indices is consistent for the children of roots, then it is consistent for whole trees of any size.

Proof. If an index-based matching scheme is consistent for the children of roots, it implies that the roots of the further subtrees-to-be-matched will be matched consistently. Inductively, the whole tree will be matched consistently. \square

Theorem 1 (Consistency — rank-based weighted bipartite matching). Consider `fixedTreeMatching` in which the applied indices are linear, the heuristic `weighted-BipartiteMatching` applied in index-to-index matching is *rank-based* (see p. 33). Then, `fixedTreeMatching` is consistent.

Proof. Positive scalar multiplication does not change the ranks of linear indices nor the ranks of the sums of linear indices. \square

Theorem 2 (Consistency — exact weighted bipartite matching, squared Euclidean cost). Consider `fixedTreeMatching` in which the indices are linear, `weighted-BipartiteMatching` is *exact*, and the vertex matching costs in (5) are $c(u, v) = \|\mathbf{i}(u) - \mathbf{i}(v)\|_2^2$ and $c'(u) = \|\mathbf{i}(u)\|_2^2$. Then, `fixedTreeMatching` is consistent.

Proof. Due to universality of linear indices (Lemma 2), we can concentrate on the problem of children-to-children matching. Basically, there can be two types of recombinations: matched children recombine among themselves, or previously unmatched children become matched (turning some unmatched children matched). However, this distinction is irrelevant since each unmatched vertex can be regarded as matched to a null vertex λ having zero indices because $c'(v) = \|\mathbf{v}\|_2^2 = \|\mathbf{v} - \mathbf{0}\|_2^2 = c(v, \lambda)$.

Assume that a vertex of T_u has two children, u and u' , which will be matched to the two children, v and v' , of some vertex in T_v . Denote the respective index vectors by \mathbf{u} , \mathbf{u}' , \mathbf{v} , and \mathbf{v}' . Assume further that T_u and T_v are multiplied by some α and β , respectively. Then

$$\begin{aligned} & c(u, v) + c(u', v') \\ &= \|\alpha \mathbf{u} - \beta \mathbf{v}\|_2^2 + \|\alpha \mathbf{u}' - \beta \mathbf{v}'\|_2^2 \\ &= \alpha^2 \mathbf{u}^\top \mathbf{u} - 2\alpha\beta \mathbf{u}^\top \mathbf{v} + \beta^2 \mathbf{v}^\top \mathbf{v} + \alpha^2 \mathbf{u}'^\top \mathbf{u}' - 2\alpha\beta \mathbf{u}'^\top \mathbf{v}' + \beta^2 \mathbf{v}'^\top \mathbf{v}' \end{aligned} \tag{18}$$

and

$$\begin{aligned}
& c(u, v') + c(u', v) \\
&= \|\alpha \mathbf{u} - \beta \mathbf{v}'\|_2^2 + \|\alpha \mathbf{u}' - \beta \mathbf{v}\|_2^2 \\
&= \alpha^2 \mathbf{u}^\top \mathbf{u} - 2\alpha\beta \mathbf{u}^\top \mathbf{v}' + \beta^2 \mathbf{v}'^\top \mathbf{v}' + \alpha^2 \mathbf{u}'^\top \mathbf{u}' - 2\alpha\beta \mathbf{u}'^\top \mathbf{v} + \beta^2 \mathbf{v}^\top \mathbf{v}
\end{aligned} \tag{19}$$

For some α and β , assume that $\{(u, v); (u', v')\}$ is the optimal matching. It means that

$$\begin{aligned}
\hat{c}(u, v) + \hat{c}(u', v') &\leq \hat{c}(u, v') + \hat{c}(u', v) \\
\Leftrightarrow -2\alpha\beta(\mathbf{u}^\top \mathbf{v} + \mathbf{u}'^\top \mathbf{v}') &\leq -2\alpha\beta(\mathbf{u}^\top \mathbf{v}' + \mathbf{u}'^\top \mathbf{v}) \\
\Leftrightarrow \mathbf{u}^\top \mathbf{v} + \mathbf{u}'^\top \mathbf{v}' &\geq \mathbf{u}^\top \mathbf{v}' + \mathbf{u}'^\top \mathbf{v}.
\end{aligned} \tag{20}$$

But (20) applies to any positive α and β , meaning that matching of two vertices is not affected by weighting. Generally, the same applies for matching vertex sets of any size: looking at (18) and (19), the last line of (20) will always consist of mixed terms, that is, inner products of matched vectors, and the inequality will hold for any number of vertices, independently from α and β . \square

Remark. `fixedTreeMatching` applying *exact* `weightedBipartiteMatching` and an L -norm as the distance function (2), is *inconsistent*. One can verify this by taking for example $\mathbf{u} = [1 \ 1]$, $\mathbf{u}' = [3 \ 2]$, $\mathbf{v} = [6 \ 1]$, and $\mathbf{v}' = [4 \ 2]$ (Fig. 9, left). Using L -norm, the optimal matching is $M = \{(u, v); (u', v')\}$, because $\hat{c}(u, v) + \hat{c}(u', v') = \|\mathbf{u} - \mathbf{v}\|_p + \|\mathbf{u}' - \mathbf{v}'\|_p = \sqrt[p]{5^p + 0^p} + \sqrt[p]{1^p + 0^p} = 6 = \sqrt[p]{3^p} + \sqrt[p]{3^p} < \sqrt[p]{3^p + 1^p} + \sqrt[p]{3^p + 1^p} = \hat{c}(u, v') + \hat{c}(u', v)$. After multiplication by two (Fig. 9, right), the optimal matching is $M = \{(u, v'); (u', v)\}$ because $\hat{c}(u, v') + \hat{c}(u', v) = \sqrt[p]{2^p + 0^p} + \sqrt[p]{0^p + 3^p} = 5 < 6 = \sqrt[p]{4^p} + \sqrt[p]{2^p} < \sqrt[p]{4^p + 1^p} + \sqrt[p]{2^p + 2^p} = \hat{c}(u, v) + \hat{c}(u', v')$.

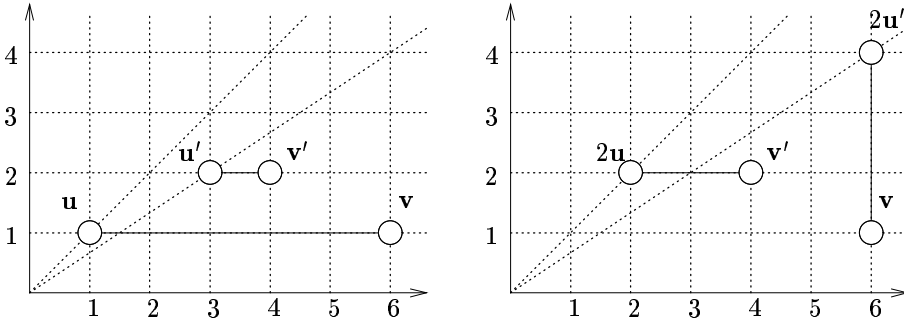


Figure 9: Inconsistency of tree matching when applying exact L -norm-based (or greedy) weighted bipartite matching. The match changes when multiplying T_u by two.

Remark. Also `fixedTreeMatching` applying *greedy* `weightedBipartiteMatching` is inconsistent. Reusing the previous example (Fig. 9), u' will be first matched to v' , leaving u for v . After multiplication by two, u will catch v' , leaving v for u' . (Looking at Fig. 9, this reasoning holds for any sensible distance function.)

Lemma 3 (Equivalence of mixtures and weighted sums). If `fixedTreeMatching` is consistent, then (14) applies: $\text{mixTrees}(T_u, T_v, \alpha) \equiv \alpha T_u + (1 - \alpha) T_v$.

Proof. `mixTrees` involves matching succeeded by weighting, while the left-hand side of (14) involves matching preceded by weighting. Consistency of matching implies, that the mutual order of weighting and matching is however insignificant. \square

Definition 14 (Consistency of mixing). Consider two attribute trees T_u and T_v , a matching scheme `fixedTreeMatching`, and the respective mixing scheme `mixTrees`. Set $T_\alpha = \text{mixTrees}(T_u, T_v, \alpha)$. Denote the matched vertices in `fixedTreeMatching`(T_u, T_v) by M . Further, denote the matched vertices in `fixedTreeMatching`(T_α, T_u) and `fixedTreeMatching`(T_α, T_v) by M_u and M_v , respectively. The mixing scheme `mixTrees` is *consistent* if

$$M_u = M_v = M \quad \forall \alpha \in [0, 1] \quad (21)$$

Thus consistency means that matching will not alter along the path of interpolation.

Lemma 4 (Convexity of indices). Consider trees T_u and T_v . Denote the respective index vectors of the roots by \mathbf{u} and \mathbf{v} , respectively. Consider $\tilde{T}_\alpha = \text{mixTrees}(T_u, T_v, \alpha)$, denoting the index vector of the root by $\tilde{\mathbf{v}} = \tilde{\mathbf{v}}(\alpha)$. Then, if the applied indices are linear (Lemma 1),

$$\tilde{\mathbf{v}} = \beta \mathbf{u} + (1 - \beta) \mathbf{v}, \quad (22)$$

where $\beta = \beta(\alpha)$ is a monotone increasing function such that $\beta(0) = 0$ and $\beta(1) = 1$.

The lemma states that the index vector of a vertex composed of two mixed vertices lies on the line segment connecting the original index vectors.

Proof. In mixing two trees T_u and T_v with a mixing coefficient α , the weights of each pair (u, v) of matched vertices become mixed as $w_\alpha = \alpha w_u + (1 - \alpha) w_v$ which is a convex sum. If the trees have attributes, also the attributes become mixed as convex sums: $\tilde{\mathbf{a}} = \beta \mathbf{a}_u + (1 - \beta) \mathbf{a}_v$, where $\beta = (\alpha w_u) / [\alpha w_u + (1 - \alpha) w_v]$. Generally, the β varies for each pair of matched vertices. A linear index associated to a vertex is a linear sum of indices of the subtree rooted to that vertex. Clearly, $\tilde{\mathbf{v}} = \mathbf{v}$ when $\alpha = 0$ and $\tilde{\mathbf{v}} = \mathbf{u}$ for $\alpha = 1$. Due to linearity indices, and the convexity of the partial sums, $\tilde{\mathbf{v}} = \beta \mathbf{u} + (1 - \beta) \mathbf{v}$. \square

Theorem 3 (Consistency of mixing / L -norm). The mixing scheme `mixTrees` using *exact* `weightedBipartiteMatching` for linear indices under L -norm is consistent.

Proof. (Triangle inequality.) Consider trees T_u and T_v . Assume that the root of T_u has D children, $U^+ = \{u_1, u_2, \dots, u_D\}$, and the root of T_v has also D children, $V^+ = \{v_1, v_2, \dots, v_D\}$. Denote the respective index vectors by $\{\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_D\}$, and $\{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_D\}$. If the child counts are initially unequal insert the required number of null children (setting the corresponding index vectors to zero). Without loss of generality we may assume that the numbering of the children corresponds to the optimal matching invoked implicitly by $\tilde{T} = \text{mixTrees}(T_u, T_v, \alpha)$. That is, the optimal matching is $M = \{(u_1, v_1); (u_2, v_2); \dots; (u_D, v_D)\}$. Consider the corresponding mixed vertices $\tilde{V}^+ = \{\tilde{v}_1, \tilde{v}_2, \dots, \tilde{v}_D\}$, and the respective index vectors $\{\tilde{\mathbf{v}}_1, \tilde{\mathbf{v}}_2, \dots, \tilde{\mathbf{v}}_D\}$ where $\tilde{\mathbf{v}}_i = \tilde{\mathbf{v}}_i(\alpha) = \beta_i(\alpha) \mathbf{u}_i + [1 - \beta_i(\alpha)] \mathbf{v}_i$ (see (13) and Lemma 4). Denote the M -consistent

matching of \tilde{V}^+ and U^+ by $M_u = \{(\tilde{v}_1, u_1); (\tilde{v}_2, u_2); \dots; (\tilde{v}_D, u_D)\}$, and respectively for V^+ by $M_v = \{(\tilde{v}_1, v_1); (\tilde{v}_2, v_2); \dots; (\tilde{v}_D, v_D)\}$ (see Fig. 10). The optimality of M means that $\text{cost}(M) = \sum_i \hat{c}(u_i, v_i) = \min$. On the other hand, when using an L -norm, by the convexity of indices (Lemma 4), $\text{cost}(M) = \text{cost}(M_v) + \text{cost}(M_u)$. Consider growing α from zero towards one. Assume that at some point α , a new matching M'_u , instead of M_u , would be optimal: $\text{cost}(M'_u) < \text{cost}(M_u)$. But then $\text{cost}(M_v) + \text{cost}(M'_u) < \text{cost}(M)$ which is a contradiction. Similarly for M_v . Clearly, the universality of matching consistency (Lemma 2) generalizes to mixing, thus only the children of the roots need to be discussed. \square

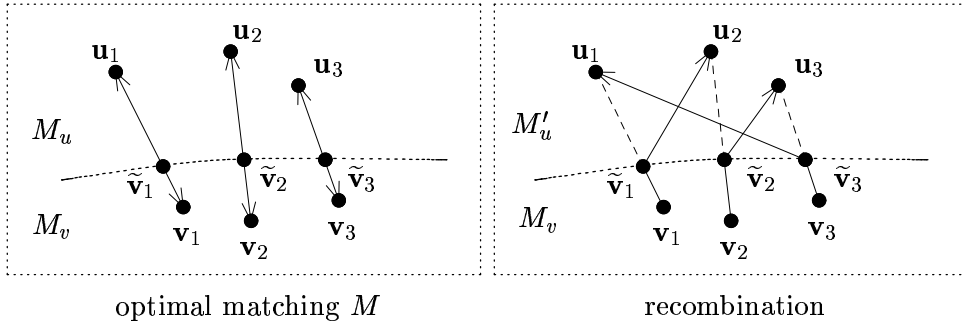


Figure 10: Consistency of heuristic tree matching applying exact index-to-index-by- L -norm matching

Remark. The mixing scheme `mixTrees` using *exact* `weightedBipartiteMatching` for linear indices under squared Euclidean distance (as in Theorem 2) is *inconsistent*. For example, consider vertices u, u', v , and v' assuming indices $\mathbf{u} = [1 \ 2]$, $\mathbf{u}' = [3 \ 1]$, $\mathbf{v} = [6 \ 2]$, and $\mathbf{v}' = [8 \ 1]$. The optimal matching is $\{(u, v); (u', v')\}$ as $(5^2 + 0^2) + (5^2 + 0^2) < (7^2 + 1^2) + (3^2 + 1^2)$ (see Fig. 11, left). A mixture of the involved trees might yield $\mathbf{v}_\alpha = 0.2\mathbf{u} + 0.8\mathbf{v} = [5 \ 2]$ and $\mathbf{v}'_\alpha = 0.8\mathbf{u}' + 0.2\mathbf{v}' = [4 \ 1]$, but then the optimal matching would be $\{(v_\alpha, v'); (v'_\alpha, v)\}$ as $(3^2 + 1^2) + (2^2 + 1^2) < (1^2 + 0^2) + (4^2 + 0^2)$ (see Fig. 11, right).

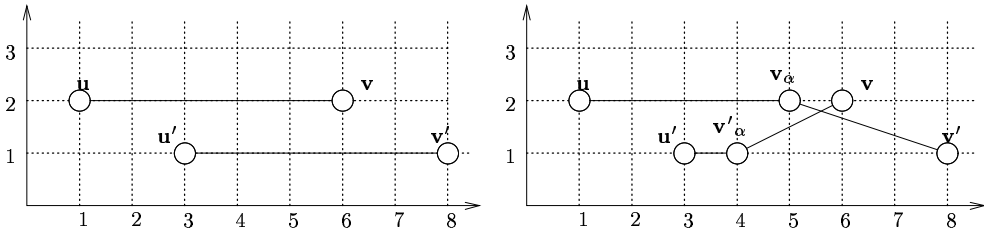


Figure 11: Inconsistency of tree mixing when applying squared L_2 -norm in weighted bipartite matching.

Remark. Mixing schemes using *rank-based* or *greedy* approach are not generally consistent. For example, the ranks change in Fig. 11: $\|\mathbf{u}\|_p < \|\mathbf{u}'\|_p$ and $\|\mathbf{v}\|_p < \|\mathbf{v}'\|_p$ but $\|\mathbf{v}_\alpha\|_p > \|\mathbf{v}'_\alpha\|_p$ for any $p \geq 1$ (at least). As to *greedy* matching, the optimal matching

would be initially $\{(u, v'); (u', v)\}$ but for example, given $\mathbf{v}_\alpha = 0.3\mathbf{u} + 0.7\mathbf{v}' = [5.9 \ 1.3]$ and $\mathbf{v}'_\alpha = 0.3\mathbf{u}' + 0.7\mathbf{v} = [5.1 \ 1.7]$, the optimal matching would be $\{(v_\alpha, v); (v'_\alpha, v')\}$.

Theorem 4 (Consistency of mixing / rank-based, equal weights). The mixing scheme `mixTrees` using *rank-based weightedBipartiteMatching* for positive linear indices, equally weighted siblings, and L_1 -norm in ranking, is consistent.

Proof. If the elements of vectors \mathbf{v} and \mathbf{u} are positive, $\|\mathbf{v}\|_1 + \|\mathbf{u}\|_1 = \|\mathbf{v} + \mathbf{u}\|_1$. In addition, for any L -norm applies $\|\beta\mathbf{v}\| = \beta\|\mathbf{v}\|$. Consider ordered sets of index vectors $\|\mathbf{u}_1\|_1 \geq \|\mathbf{u}_2\|_1 \geq \dots \geq \|\mathbf{u}_D\|_1$ and $\|\mathbf{v}_1\|_1 \geq \|\mathbf{v}_2\|_1 \geq \dots \geq \|\mathbf{v}_D\|_1$. The inequalities can be multiplied by β and $1 - \beta$, and summed, yielding $\|\beta\mathbf{u}_1 + (1 - \beta)\mathbf{v}_1\|_1 \geq \|\beta\mathbf{u}_2 + (1 - \beta)\mathbf{v}_2\|_1 \geq \dots \geq \|\beta\mathbf{u}_D + (1 - \beta)\mathbf{v}_D\|_1$. Due to universality of matching consistency, the theorem follows. \square

Theorem 5 (Linearity of mixtures). Consider mixing scheme $\tilde{T}(\alpha) = \text{mixTrees}(T_u, T_v, \alpha)$ in which weights are mixed as $\tilde{w} = w_u + w_v - w_u w_v / [\alpha w_u + (1 - \alpha)w_v]$ (instead of $\tilde{w} = \alpha w_u + (1 - \alpha)w_v$ in (13)). If `mixTrees` is (still) consistent, and the vertex cost $c(u, v)$ applied in approximative overall distance function $\widehat{\text{dist}}(T_u, T_v)$ is defined as in Table 2, then

$$\widehat{\text{dist}}(T_u, T_\alpha) + \widehat{\text{dist}}(T_\alpha, T_v) = \widehat{\text{dist}}(T_u, T_v). \quad (23)$$

The theorem says that the described mixture, seen by the approximative distance function, lies on the line segment joining the matched trees.

Proof. Consider mixing T_u and T_v . Assume that $u \in U$ and $v \in V$ become (implicitly) matched in $\tilde{T} = \text{mixTrees}(T_u, T_v, \alpha)$, and denote the resulting vertex in \tilde{T} by \tilde{v} . The respective attribute becomes $\tilde{\mathbf{a}} = \beta\mathbf{a}_u + (1 - \beta)\mathbf{a}_v$, where $\beta = \frac{\alpha w_u}{\alpha w_u + (1 - \alpha)w_v}$, and weight becomes $\tilde{w} = w_u + w_v - w_u w_v / [\alpha w_u + (1 - \alpha)w_v]$. (If \tilde{v} corresponds to an unmatched vertex, the above mixing rules can be used by setting the weight of the missing vertex to zero.) The cost of matching \tilde{v} to v is

$$\begin{aligned} c(\tilde{v}, v) &= \frac{\tilde{w} + w_v}{2} \|\tilde{\mathbf{a}} - \mathbf{a}_v\| = \frac{\tilde{w} + w_v}{2} \|\beta\mathbf{a}_u + (1 - \beta)\mathbf{a}_v - \mathbf{a}_v\| \\ &= \frac{\tilde{w} + w_v}{2} \frac{\alpha w_u}{\alpha w_u + (1 - \alpha)w_v} \|\mathbf{a}_u - \mathbf{a}_v\|, \end{aligned} \quad (24)$$

and respectively for matching \tilde{v} to u ,

$$\begin{aligned} c(\tilde{v}, u) &= \frac{\tilde{w} + w_u}{2} \|\tilde{\mathbf{a}} - \mathbf{a}_u\| = \frac{\tilde{w} + w_u}{2} \|\beta\mathbf{a}_u + (1 - \beta)\mathbf{a}_v - \mathbf{a}_u\| \\ &= \frac{\tilde{w} + w_u}{2} \frac{(1 - \alpha)w_v}{\alpha w_u + (1 - \alpha)w_v} \|\mathbf{a}_u - \mathbf{a}_v\| \end{aligned} \quad (25)$$

thus adding up to

$$c(\tilde{v}, v) + c(\tilde{v}, u) = \frac{(\tilde{w} + w_v)\alpha w_u + (\tilde{w} + w_u)(1 - \alpha)w_v}{2(\alpha w_u + (1 - \alpha)w_v)} \|\mathbf{a}_u - \mathbf{a}_v\| \quad (26)$$

$$\begin{aligned}
&= \frac{\tilde{w}(\alpha w_u + (1 - \alpha)w_v) + w_u w_v}{2(\alpha w_u + (1 - \alpha)w_v)} \|\mathbf{a}_u - \mathbf{a}_v\| & (27) \\
&= \frac{1}{2} \left(\tilde{w} + \frac{w_u w_v}{\alpha w_u + (1 - \alpha)w_v} \right) \|\mathbf{a}_u - \mathbf{a}_v\| \\
&= \frac{1}{2}(w_u + w_v) \|\mathbf{a}_u - \mathbf{a}_v\| = c(u, v).
\end{aligned}$$

Due to mixing consistency, vertex-to-vertex matches are stable, and since the above summation applies to every vertex, the theorem follows. \square

Remark. Using the revised weighting rule makes Lemma 1 inapplicable, which turns also Lemma 4 inapplicable. Only the consistency of ranked matching, Theorem 4, holds but it implies that the mixture curve of weights is the same among siblings.

Conclusions. A collection of lemmas and theorems was presented, exposing stability aspects of heuristic matching and mixing. Among the discussed index-to-index distance functions, there seems to be no universal choice implying consistency in both matching and mixing as well as linearity of mixtures. Above these ideal properties, one can expect that the consistency of mixing should be the most significant: once the learning medium has survived initialization and a possibly noisy overall adaptation stage, it is crucial that the prototypes “get locked to their targets.” Among the alternative approaches for WEIGHTED BIPARTITE MATCHING, the L -norm-based exact procedure seems the safest. However, the rank-based matching is computationally more attractive especially if the index set is small, and the indices are positive (Theorem 4). It seems that the metrics must be chosen separately for each application, compromising between consistency and linearity. To conclude, among the cases shown in Fig. 8, the one closest to the practical implementations of index-based matching seems to be the case e), which should be satisfactory for most applications.

5.4 The self-organizing map of attribute trees

The analysis presented in Secs. 5.1–5.3 applies especially to learning processes involving trees as smoothly adaptive object models. This section presents such a scheme, summarizing several topics addressed also in the publications of this thesis.

The *Self-Organizing Map* (SOM) invented by Kohonen (1982, 1990, 1995) is one of the most popular neural network methods in the world. It has been applied in various recognition, prediction and monitoring problems (Kohonen et al. 1996). Like all the neural networks, the Self-Organizing Map is an adaptive system that can *learn* from examples. A special property of the SOM is that the training stage is unsupervised.

Formally, the SOM consists of an array of learning units, *neurons*. During training the neurons smoothly adapt to presented data. By default, the data objects and the learning objects associated with the neurons are of the same type, say real-valued vectors. In principle, the studied objects could be anything, as far as a *distance function* and an

adjustment operation are defined. A learning step consists of presenting a sample to the map, finding the most resembling neuron — the best-matching unit — and adjusting that neuron *and its neighbors* towards the sample. The radius of the neighborhood decreases gradually during training. As a result, the neurons represent the input data in an *orderly fashion*: nearby neurons correspond to nearby locations in the input space. In other words, the SOM implements a topology-preserving mapping.

If the SOM is used in recognition or monitoring tasks, the map is calibrated by attaching labels describing, say, classes to be recognized. In Fig. 12, we outline an a potential application based on the weather radar data: a “weather map”. In Fig. 12, the underlying hexagonal array illustrates the internal data presentation applied by the computer, the suns and the clouds are labels attached by an expert (radar meteorologist), and the trajectory illustrates the automated, operational stage of monitoring the development of weather, or to be more specific, sky conditions.

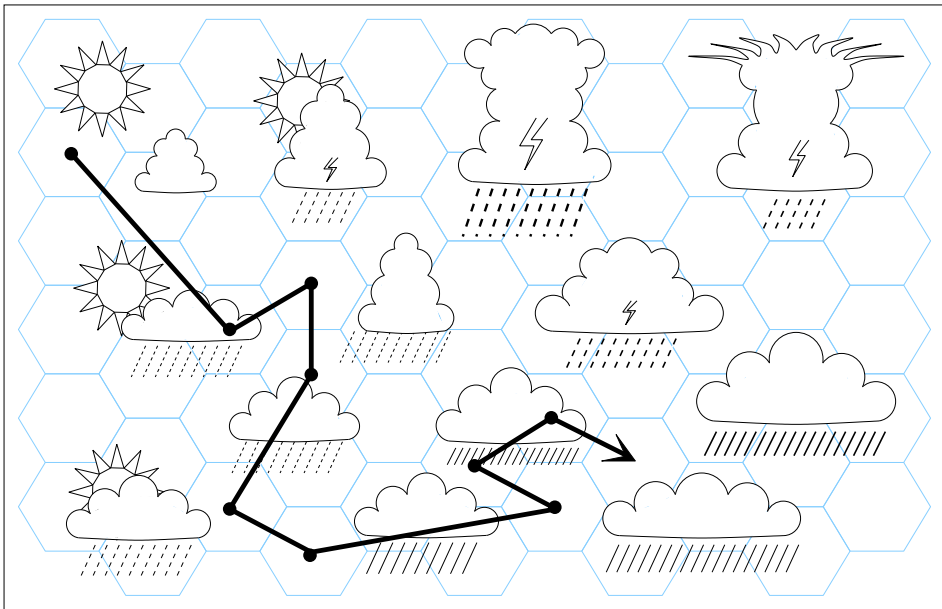


Figure 12: A hypothetical application, “weather map”. The trajectory indicates the sky conditions as a function of time.

The original and most popular version of the SOM is a two-dimensional array of vectors. By default, the applied distance function is Euclidean, and the adjustment of neurons is realized as interpolation between samples and map vectors. Also alternative topologies and data structures have been suggested. Conical or annular topology is preferred in applications involving circularities in input space. The Tree-Structured Self-Organizing Map (TS-SOM) by Koikkalainen (1994) has the form of a quadtree: the map is recursively divided into two-by-two maps. The tree-like structure implies speed-up in both training and operation. However, it must be noted that the trees discussed in this thesis are data objects, not algorithmic structures as in the TS-SOM.

As mentioned above, the crucial point is that a distance function and an adjustment operation are defined for the objects involved. In addition, the computational complexity must be taken into account as practical constraint. For example, assume that the training stage of a SOM involves M data samples and N map units. At each training step, a sample is presented to the map, and the search for the best-matching unit requires N distance evaluations, followed by N adjustment operations. The sample set is often used over and over, hence the complexity of distance evaluation and adjustment is critical for the overall computational effort. For a continuous object representation, the adjustment operation is ideally based on the gradient of the distance function, but such a gradient is not always available. Also discrete objects can be considered; essentially, the point is that the adjustment should tune objects of the map towards the presented samples. If an adjustment operator cannot be defined explicitly, it can be derived indirectly from the distances. Kohonen (1985) showed how medians of symbol strings can be calculated, further yielding the Self-Organizing Map of Symbol Strings (Kohonen 1996, Kohonen and Somervuo 1998) which can be used for example in clustering protein sequences (Somervuo and Kohonen 2000).

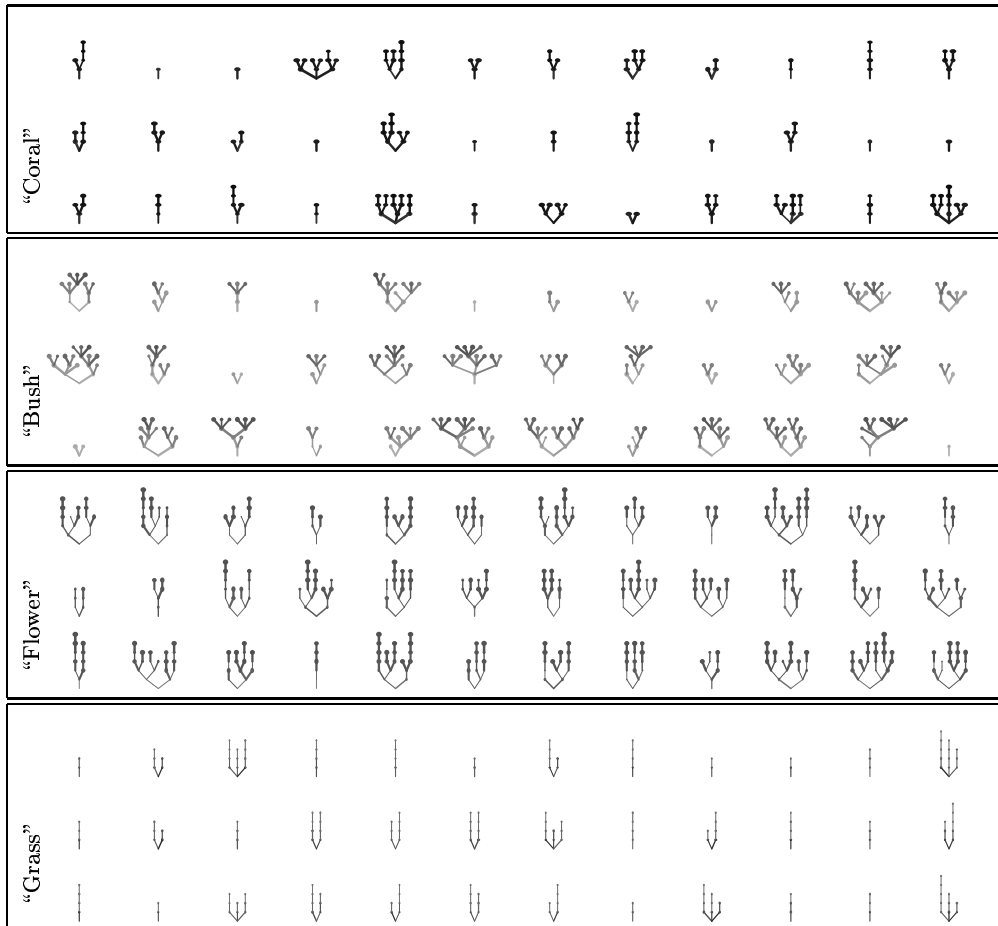
The central innovation in this thesis is the Self-Organizing Map of Trees (SOM-T, Publ. III) and more generally, the Self-Organizing Map of Attribute Trees (SOM-AT, Publ. VI). Given the proposed tree distance function (6) and mixing rule (13), implementing the SOM-T or the SOM-AT is straightforward. Practically, no tree-specific modifications are needed in the basic algorithm: the required tree operations can be linked to the code as portable modules. The only exception is *pruning*: the trees in the map have to be pruned regularly in order to avoid generation of massive unions of trees. Pruning means deleting subtrees having weights that fall below a predefined limit.

The SOM-AT is suited for graphical illustrations. Let us consider four sets of artificial trees, each set consisting of hundred trees. Three vertex attributes are applied: vertex size, intensity and elongation. The trees within a set resemble each other, so the sets represent four “species”. Samples of each species are shown in Table 6. A map trained with this data is shown in Fig. 13.

Although there is no one-to-one analogy between trees and vectors, consistency of tree matching, mixing and distance (Sec. 5.3) implies that the behaviour of the SOM-AT can be expected to be close to that of the standard SOM. Occasional inconsistencies and the lack of certain properties (like associativity) may effect training, but perhaps only by increasing the processing time. One may assume that the errors resulting from heuristic matching behave as noise. It should be remembered that noise is sometimes a desired ingredient in training a neural network — such as the SOM — as it may accelerate learning and also generalize the result (Bishop 1995).

As explained in Publications III and VI, the SOM-AT involves tunable parameters, most of which are however inherited from the standard SOM. It seems that the overall implementation of tree matching, mixing and distance functions is more significant in practice. Also, the success of self-organization depends more on the dimensionality of the studied

Table 6: Four artificial attribute tree “species”.



phenomenon than on the applied data structure, as pointed out in Publ. III. For example, an application may produce apparently complex trees while the space spanned by the trees is essentially two dimensional: each tree may correspond to a point in a plane. The practical assumption here is that determining such mappings — that is, optimal indices — by observing tree distributions is generally nontrivial. For example, it would be hard to define an explicit mapping from the trees shown in Table 6 to two dimensions.

To conclude, we claim that when processing tree data with the SOM, a matching-based approach — even using rather simple indices — is generally applicable, hence designing complicated, application-dependent tree-to-vector mappings will not often pay the effort. In addition, the matching approach allows using similar objects (trees) both as input and as learning medium, which facilitates controlling the learning process. In the next section, the former claim will be motivated more theoretically.

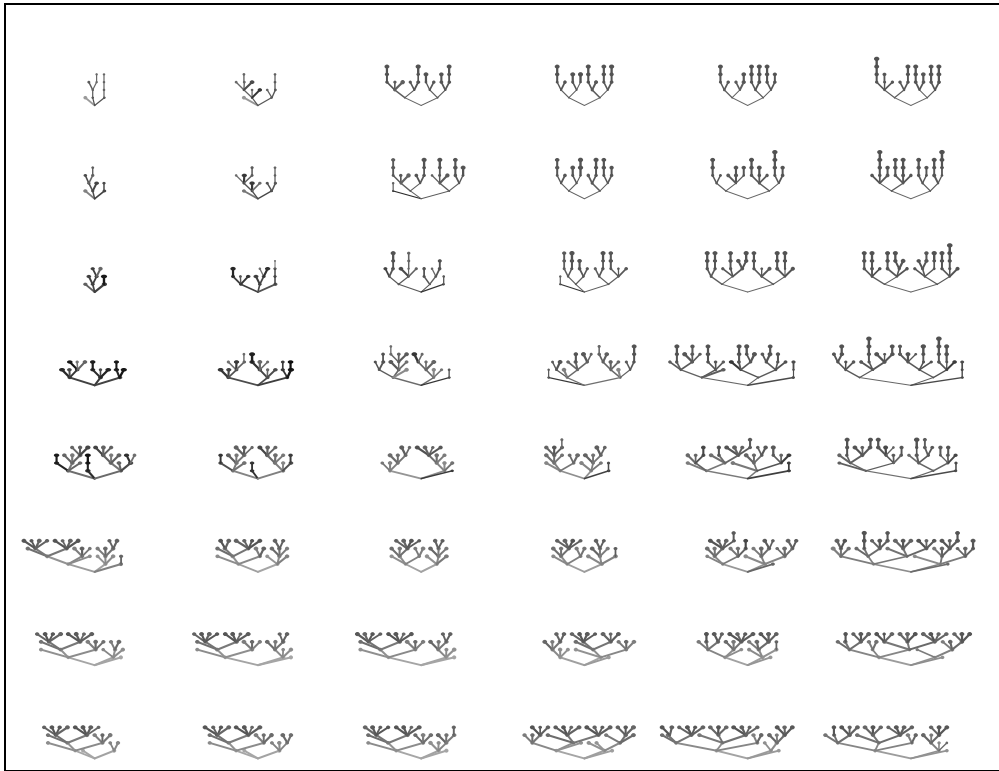


Figure 13: A Self-Organizing Map of Attribute Trees trained with artificial tree “species”. The displayed trees are not labels on a map but the true contents of the map. Essentially, the trees *are* the map.

6 Bounds of matching performance

Heuristic methods are motivated by savings in computational effort. As a trade-off, they produce suboptimal results. The computational complexity of the proposed algorithm (Alg. 2) was discussed in Sec. 4.1, but nothing has been stated about the performance in terms of matching success, that is, the number or the proportion of correctly matched vertices. Based on simple tree distributions, this section presents some previously unpublished bounds of performance. For simplicity of formulae, trees without weights and other attributes will be considered, and *fit*, number of matched vertices will be discussed instead of distance.

6.1 Indexing, index-based matching or exact matching?

Indices proposed in Sec. 4.2 have been mainly designed for index-based matching. However, since indices are descriptions of trees, they can be used as such for comparing trees. Intuitively, if the applied indices provide no information, no sensible matchings will be obtained either. On the other hand, a 100% success in index-based matching would mean that the applied indices carry complete information about trees — in which case matching would be an unnecessary effort. Hence, it is natural to ask whether the similarity of two trees can be any better approximated through index-based matching than by comparing index vectors directly using, say, Euclidean distance.

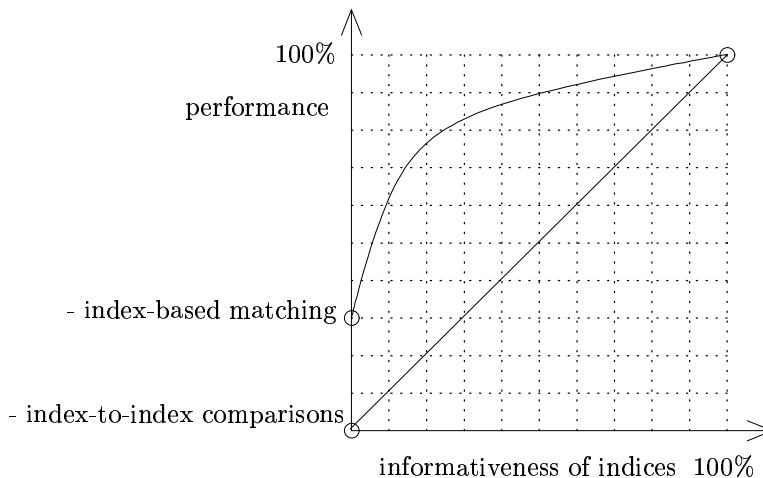


Figure 14: Hypothetical performance of distance approximation.

In fact, the unstated hypothesis beyond index-based matching discussed in Sec. 4.1 is that index-based matching leads to better distance approximations, unless the indices provide complete information. The hypothesis is illustrated in Fig. 14.

In the following, the validity of the hypothesis is verified using a simple distribution of random trees. A lower bound of index-based matching is derived theoretically for the worst possible indices, random indices. Further, it will be shown that the proposed heuristic matching scheme is reliable with respect to the required computational effort. The two types of problems to be addressed in parallel are *matching of random trees* and *matching of isomorphic trees*.

The applied notations are listed in Table 7. The notations equipped with an asterisk (*) apply to whole trees; without asterisk to distinct vertices. All the distributions and expectations are functions of \overline{D} and \overline{H} , which are however omitted from notations for convenience. Also bracketed operands are omitted when appropriate.

Table 7: Applied notations.

d, \overline{D}	child count, maximum child count of a vertex
h, \overline{H}	height, maximum height of a tree
$P(v)$	probability of the existence of a vertex
$P(v_1 v_2)$	probability of the existence of a vertex v_1 given that v_2 exists
$P(d)$	probability for a vertex to have d children
$E[d]$	expectation of child count of a vertex
$E^*[n]$	expectation of vertex count of a tree
$E_r[m]$	expectation of matches when matching randomly the children of two random vertices
$E_r^*[m]$	expectation of matches when matching randomly two random trees
$E_p[m d]$	expectation of the number of unchanged positions in a random permutation of a set of d items (here: children)
$E_p[m]$	expectation of the number of unchanged positions in random permutations of up to \overline{D} children
$E_i^*[m]$	expectation of the number of matched vertices when matching randomly two isomorphic trees

Definition 15 (coinflip tree). A coinflip tree is a random tree which has at least one vertex — the root, v_1 — and in which any vertex v_s has children $v_{s,1}, v_{s,2}, v_{s,3}\dots$ according to the probability distribution

$$P(v_{s,1}|v_s) = \frac{1}{2}, \quad (28)$$

$$P(v_{s,[i+1]}|v_{s,i}) = \frac{1}{2}. \quad (29)$$

Thus, a coinflip tree can be thought as constructed by flipping a coin: for each vertex, children are added one at a time as long as the result is “heads”; the first “tails” terminates the process. Consequently, any tree of N vertices can be presented by a binary string consisting of N zeros and N ones, corresponding to a sequence of coinflips. In

number theory, the integers (ranks) associated with these binary strings are called *Catalan numbers*. The average height of ordered trees (see de Bruijn et al. 1972) as well as ranking of ordered trees (Kreher and Stinson 1999) have been studied by regarding trees as Catalan numbers. This encoding is also addressed in gambling theory (fifty-fifty betting) or in discrete random walks in a straight line.

Clearly, the resulting distribution of child count is geometric: $P(d) = 1/2^{d+1}$. The procedure is repeated recursively for each child. This probabilistic model for generating trees is illustrated in Fig. 15. The line thickness corresponds to the probability. It should be emphasized that while the generation of coinflip trees obeys an order, the constructed trees are nevertheless regarded unordered.

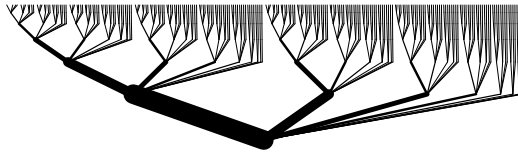


Figure 15: An illustration of the coinflip tree model.

Definition 16 (constrained coinflip tree). A constrained coinflip tree is a coinflip tree with maximum height \overline{H} and maximum child count \overline{D} . Formally:

$$P(v_{s.1}|v_s) = \begin{cases} \frac{1}{2}, & \text{if } \Delta(v_s) < \overline{H}, \\ 0, & \text{if } \Delta(v_s) = \overline{H}, \end{cases} \quad (30)$$

$$P(v_{s.(d+1)}|v_{s.d}) = \begin{cases} \frac{1}{2}, & \text{if } d < \overline{D}, \\ 0, & \text{if } d = \overline{D}, \end{cases} \quad (31)$$

where $\Delta(v_s)$ is the depth of vertex v_s . The children are distributed as above with the exception $P(\overline{D}) = P(\overline{D} - 1)$. Any constrained coinflip tree can be thought of first generating the tree by the unconstrained model but then *truncating* the branches exceeding the constraints \overline{H} and \overline{D} . A slightly different distribution would be obtained by *discarding* the trees violating the constraints, but the first policy is more convenient for the considerations presented in this section.

Evidently, the coinflip model is one of the simplest possible tree distributions. The simplicity of definition means also that the derivation of expectations is easy. As a further attractive property, *any tree* can be regarded as an instance of the (unconstrained) coinflip tree model, suggesting that a result derived for coinflip trees can be generalized for *arbitrary* tree distribution by appropriate weighting. The constrained coinflip distribution is also feasible in generating samples for experiments: the expected child count is location invariant and finite. Further, the expected total vertex count is finite for any *finite* \overline{D} .

If a vertex is allowed to have maximally \overline{D} children, the expected child count is

$$E[d] = \sum_{i=0}^{\overline{D}} iP(i) = \sum_{i=0}^{\overline{D}-1} i \left(\frac{1}{2}\right)^{i+1} + \overline{D} \left(\frac{1}{2}\right)^{\overline{D}} = 1 - (1/2)^{\overline{D}} \quad (32)$$

using $\sum_{n=0}^N nq^n = q(1 - q^N)/(1 - q)^2 - (Nq^{N+1})/(1 - q)$. Values of $E[d]$ for some \overline{D} are listed in Table 8.

Table 8: Expectation of child count $E[d]$ assuming maximal child count \overline{D} .

\overline{D}	0	1	2	3	4	5	∞
$E[d]$	0	1/2	3/4	7/8	15/16	63/64	1

By location invariance, the d 's of subsequent generations are independent. With finite \overline{H} , the expected vertex count of a tree is the geometric sum of $E[d]$:

$$E^*[n] = \sum_{h=0}^{\overline{H}} E[d]^h = \frac{1 - [1 - (1/2)^{\overline{D}}]^{\overline{H}+1}}{1 - [1 - (1/2)^{\overline{D}}]} = 2^{\overline{D}} \left(1 - [1 - (1/2)^{\overline{D}}]^{\overline{H}+1}\right). \quad (33)$$

Values of $E^*[n]$ for some \overline{D} and \overline{H} are listed in Table 9. A sample distribution for $\overline{D} = 5$ and $\overline{H} = 5$ is shown in Fig. 26, Appendix A.3.

Table 9: Expectation of vertex count $E^*[n]$ for some \overline{H} and \overline{D} .

$\overline{H} \backslash \overline{D}$	0	1	2	3	4	5	∞
0	1	1	1	1	1	1	1
1	1	1.5	1.75	1.88	1.94	1.97	2
2	1	1.75	2.31	2.64	2.82	2.91	3
3	1	1.88	2.73	3.31	3.64	3.82	4
4	1	1.94	3.05	3.90	4.41	4.70	5
5	1	1.97	3.29	4.41	5.14	5.55	6
∞	1	2	4	8	16	32	

6.2 Lower bounds for matching — performance of random indices

Matching coinflip trees randomly. Let us consider matching two trees generated by the same coinflip tree distribution constrained by fixed \overline{H} and \overline{D} . In the proposed matching scheme, subtrees are recursively matched according to their indices. Random indices are clearly the worst possible indices: matching with random indices equals random matching. However, despite the noninformativeness of random indices, there is some latent topological information remaining: the child counts, d_1 and d_2 . In 1-to-1 matching, $\min(d_1, d_2)$ children become matched — and this is the reason why random

matching leads to a better approximation of tree-to-tree similarity than that obtained directly by comparing indices globally. Thus, the key is the expectation of $\min(d_1, d_2)$, the expectation of “matched” branches, which according to (39) in Appendix A.1 is

$$E_r[m] = \frac{1 - (1/4)^{\overline{D}}}{3}. \quad (34)$$

The values of $E_r[m]$ for some \overline{D} are shown in Table 10.

Table 10: Expectation of matches $E_r[m]$ as a function of maximal child count \overline{D} .

\overline{D}	0	1	2	3	4	5	∞
$E_r[m]$	0	1/4	5/16	21/64	85/256	341/1024	1/3

Thus, when matching two random trees randomly, the roots become always matched and approximately $E_r[m]$ children of the root become matched. According to the coinflip model (Def. 15), each of these $E_r[m]$ matched children has the *same distribution of children*, $E[d]$, and hence the *same expectation of matched children*: $E_r[m]$. Thus the expectation for the number of root’s matched grandchildren is $E_r^2[m]$, and $E_r^n[m]$ for n th generation. The expectation of total number of matched vertices in trees of maximal height \overline{H} and maximal child count \overline{D} is

$$E_r^*[m] = \sum_{h=0}^{\overline{H}} E_r^h[m] = \frac{1 - E_r[m]^{\overline{H}+1}}{1 - E_r[m]}. \quad (35)$$

For example, $E_r^*[m] = 5/4$ when $\overline{D} = 1$ and $\overline{H} = 1$, and $E_r^*[m] \rightarrow 3/2$ when \overline{D} and \overline{H} approach infinity. Some values of $E_r^*[m]$ are listed in Table 11.

Table 11: Expectation of matches $E_r^*[m]$ when matching two coinflip trees randomly.

$\overline{H} \backslash \overline{D}$	0	1	2	3	4	5	∞
0	1	1	1	1	1	1	1
1	1	1.25	1.31	1.33	1.33	1.33	1.33
2	1	1.31	1.41	1.44	1.44	1.44	1.44
3	1	1.33	1.44	1.47	1.48	1.48	1.48
4	1	1.33	1.45	1.48	1.49	1.49	1.49
5	1	1.33	1.45	1.49	1.50	1.50	1.50
∞	1	1.33	1.45	1.49	1.50	1.50	$1\frac{1}{2}$

Matching isomorphic coinflip trees randomly. In many applications, similarity of two objects is of interest only if the objects are already known to be at least coarsely similar. For example, in recognizing trees one could first compare the index vectors of the sample against those of the prototype trees and take into further consideration only those trees of which the index vectors are similar enough. Hence, sometimes the accuracy

of small distances is essential while large distances are not considered at all. Thus, it is worthwhile to study how random matching copes with matching of isomorphic trees.

When matching two isomorphic trees randomly, the first step consists of “matching” the d children of one root to a random permutation of the d respective children of the other root. There are $d!$ possible permutations. Observing a single position, the probability of a correct match is $1/d$, thus for a given vertex having d children the expectation is

$$E_p[m|d] = \begin{cases} 0 & \text{if } d = 0, \\ d \cdot 1/d \equiv 1 & \text{if } d > 0. \end{cases} \quad (36)$$

Thus, if there are any children, one child is expected to become matched with its true isomorphic counterpart while the other children become matched to separate counterparts. The expectation of the correct matches over all the possible numbers of children up to \overline{D} is

$$E_p[m] = \sum_{d=0}^{\overline{D}} E_p[m|d]P(d) = \sum_{d=1}^{\overline{D}} P(d) = 1 - P(0) = 1 - \frac{1}{2} = \frac{1}{2}. \quad (37)$$

The expectation of *incorrect* matches is

$$E'_p[m] = E[d] - E_p[m] = 1 - (1/2)\overline{D} - 1/2 = 1/2 - (1/2)\overline{D}. \quad (38)$$

Notice that only (38) assumes the coinflip distribution; (37) assumes $P(0) = 1/2$ and (36) applies generally. To a large extent, also the following discussion applies to any location invariant random tree distributions, but let us keep in mind that the overall framework is the coinflip distribution.

The matching can be thought as a procedure which starts by a definite isomorphism (the roots) but is progressively taken over by false random choices when proceeding upwards in the trees. The result of such a procedure contains two types of successful matches: a tree of randomly preserved true isomorphisms as well as deviations from that tree containing still some matches due to occasional similarity. The isomorphism-retrieving part consists of subsequent generations of $E_p[m]$ children whereas the randomly matching parts consist of by-chance matching subsequent generations of $E_r[m]$ children. Each of the latter type is initiated by $E'_p[m]$ children mismatched in a stage of the isomorphic chain. The *success tree* depicted in Fig. 16 shows the geometry of the isomorphic-preserving part (solid line) and the by-chance matching parts (dotted line).

Given a tree distribution, the expectation of matched vertices when randomly matching two isomorphic trees is the sum of the cumulative products determined by all the paths starting from the root. Vertex count of a by-chance matching path of length h has expectation $E'_p E_r^*(h) = E'_p \sum_{n=0}^{h-1} E_r^n = E'_p (1 - E_r^h)/(1 - E_r) = E'_p/(1 - E_r)(1 - E_r^h)$. The total expectation consists of the sum of the by-chance paths and the sum of the

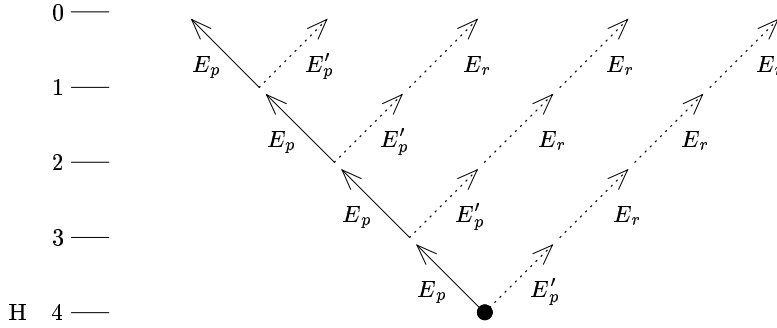


Figure 16: Illustration of the expectations related to random matching of isomorphic trees.

isomorphic path.

$$\begin{aligned}
 E_i^*[m] &= \sum_{h=0}^{\bar{H}-1} E_p^h E'_p E_r^* (\bar{H} - h) + \sum_{h=0}^{\bar{H}} E_p^h \\
 &= \frac{E'_p}{1 - E_r} \sum_{h=0}^{\bar{H}-1} E_p^h - \frac{E'_p}{1 - E_r} \sum_{h=0}^{\bar{H}-1} E_p^h E_r^{\bar{H}-h} + \frac{1 - E_p^{\bar{H}+1}}{1 - E_p} \\
 &= \frac{E'_p}{1 - E_r} \left[\frac{1 - E_p^{\bar{H}}}{1 - E_p} \right] - \frac{E_r^{\bar{H}} E'_p}{1 - E_r} \left[\frac{1 - E_p/E_r^{\bar{H}}}{1 - E_p/E_r} \right] + \frac{1 - E_p^{\bar{H}+1}}{1 - E_p} \\
 &= \frac{E - E_p}{1 - E_r} \left[\frac{1 - E_p^{\bar{H}}}{1 - E_p} - \frac{E_r^{\bar{H}} - E_p^{\bar{H}}}{1 - E_p/E_r} \right] + \frac{1 - E_p^{\bar{H}+1}}{1 - E_p}.
 \end{aligned}$$

Some values of $E_i^*[m]$ are listed in Table 12.

Table 12: Expectation of matches $E_i^*[m]$ when matching two isomorphic trees.

$\bar{H} \backslash N$	0	1	2	3	4	5	∞
0	1	1	1	1	1	1	1
1	1	1.33	1.57	1.73	1.84	1.90	2
2	1	1.44	1.85	2.23	2.35	2.48	2.67
3	1	1.48	1.97	2.55	2.63	2.80	3.06
4	1	1.49	2.02	2.75	2.77	2.97	3.27
5	1	1.50	2.05	2.88	2.84	3.06	3.38
∞	1	1.50	2.06	3.11	2.91	3.15	$3\frac{1}{2}$

Conclusions. Let us first consider random matching of two random trees. Studying Fig. 14, Table 9, and Table 11 leads to the following conclusion. When matching coinflip trees randomly, up to $1\frac{1}{2}$ vertices are expected to match. At first sight, the expectation seems small as roots contribute always by one match but, for example, 50% of the coinflip trees are single-vertex trees, thus in 75% of matched trees the number of matched vertices will not exceed one. It is more sensible to consider the proportion of matched vertices, $E_r^*[m]/E^*[d]$, keeping in mind that it is yet not the actual success rate. For example

for trees constrained by $\overline{D} \leq 5$ and $\overline{H} \leq 5$, that proportion is at least $1\frac{1}{2}/5.55 \approx 25\%$. Hence, random matching succeeds in matching *some* vertices, and because the number of matched vertices $E_r^*[m]$ is bounded by the size of the smaller tree, that number must have positive correlation with the correct number obtained by exact matching.

When matching randomly two isomorphic trees, the optimal number of matched vertices is known — as it simply equals the size of the trees. Thus for $\overline{D} \leq 5$ and $\overline{H} \leq 5$, the actual success rate is around $3.06/5.55 \approx 55\%$.

The bounds derived above suggest that random matching can be used in approximating coarsely a tree-to-tree distance, $\text{dist}(T_1, T_2)$, whereas the direct correlation between random index vectors is always zero and hence useless.

6.3 Experimental performance of the proposed indices

As shown above, the theoretical lower bounds for matching can be derived by studying the performance of the worst possible matching, random matching. Thus, any indexing scheme should imply better matching but it is however difficult to derive theoretical bounds for the performance of the indices proposed in this thesis (Sec. 4.2). In the following, the power of the indices is studied experimentally. In the experiments, `fixedTreeMatching` (Alg. 2) applied *maximal matching* (p. 33) in `weightedBipartiteMatching` of indices.

Performance of different index combinations. The five most central indices discussed in this thesis are listed in Table 13. As there are 32 possible combinations that can be composed of five indices, each combination corresponds to a binary number of five bits. Let 2^j be the marker of j th index. Then, an index set can be denoted by \mathbf{i}_b , where b is the sum of markers of the indices included. The empty set \mathbf{i}_0 corresponds to random indexing scheme.

Table 13: The five most central indices.

index	marker
D child count	1
S descendant count	2
H height	4
C center of mass	8
σ standard deviation of child count	16

The data of this experiment consisted of 1000 coinflip trees which had been generated under maximal child count $\overline{D} = 5$ and maximal height $\overline{H} = 5$. The trees are displayed in Fig. 26, Appendix A.3. Theoretically, the average vertex count of this constrained distribution is 5.55 vertices (Table 9). Exactly half of the trees consist of a single vertex in both the unconstrained and the constrained coinflip distribution. In the generated set (Fig. 26), the average size is 5.2 vertices and 504 of out the 1000 trees are singular vertices, which serves as the first check for the distribution. Further validation can be

carried out by comparing the obtained distribution against the theoretical distribution, $P(n|5,5)$, in Fig. 27, Appendix A.3. In fact, the dataset can be considered to present the unconstrained distribution $P(n)$ equally well.

First, all the 1000 trees were matched against another randomly chosen tree in the set. The results of matching are shown in Fig. 17. According to (35) and Table 11, with $\overline{H} = 5$ and $\overline{D} = 5$ the theoretical minimum is $E_r^*[m] = 1.50$. The best performance (1.701/1.704=99.8%) is obtained by the combinations $\mathbf{i}_{10} = [S, C]$, $\mathbf{i}_{14} = [S, H, C]$, $\mathbf{i}_{26} = [S, C, \sigma]$, and $\mathbf{i}_{30} = [S, H, C, \sigma]$. The most powerful single index seems to be $\mathbf{i}_2 = S$ (99.6%), while the least powerful one is $\mathbf{i}_1 = D$ (97.4%).

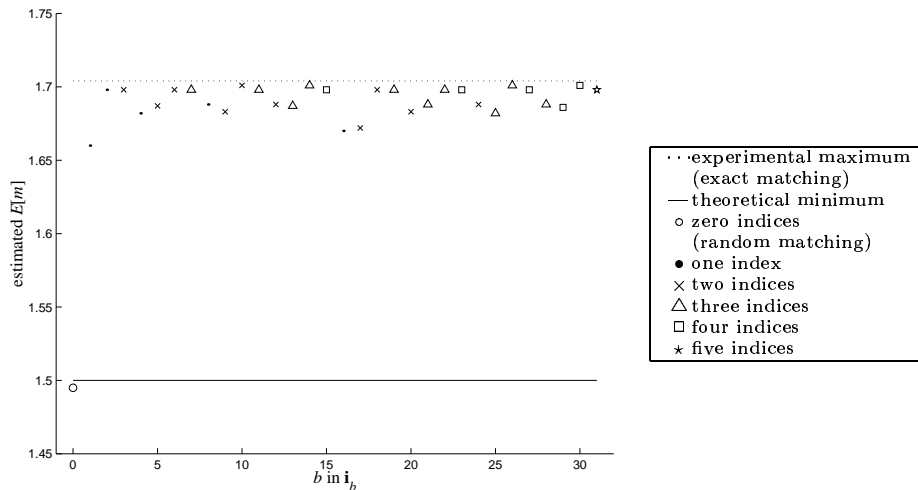


Figure 17: Matching performance measured as average number of matched vertices for two random coinflip trees ($\overline{D} = 5, \overline{H} = 5$) using random matching (\mathbf{i}_0) and all the combinations ($\mathbf{i}_1, \mathbf{i}_2, \dots, \mathbf{i}_{31}$) of indices D, S, H, C , and σ .

Each of the 1000 trees was also matched against a random isomorphic counterpart, that is, to a randomly hashed copy of the tree. The results of the isomorphic matching were impeccable. Even the worst index, child count (D), found 99.7% of the possible matches. Plain random matching found $3.10/5.21 = 0.60\%$ of the possible matches, which is close to the theoretical value (3.06, Table 12).

Performance of the indices as a function of the tree size. Generally, large trees can be expected to be more complex than small ones, but using the proposed scheme (Alg. 2) the number of indices — “the width of the information channel” — is fixed. Thus, one can expect that the errors in matching large trees are progressively larger. One can also expect that the most fatal confusions are made in matching the children of the roots. Next, let us study how matching performance changes as a function of the vertex count of trees.

First, we generated the complete sets of the unordered trees of maximally ten vertices. The number of trees in the respective sets is 1, 1, 2, 4, 9, 20, 48, 115, 286, and 719, adding

up to 1205 trees. The experiment involved matching all the pairs within each set. The matching was carried out both exactly and by using index-based approach employing the best and worst single indices, $i_1 = [D]$ and $i_2 = [S]$ (according to Fig. 17). The results are shown in Fig. 18 (left).

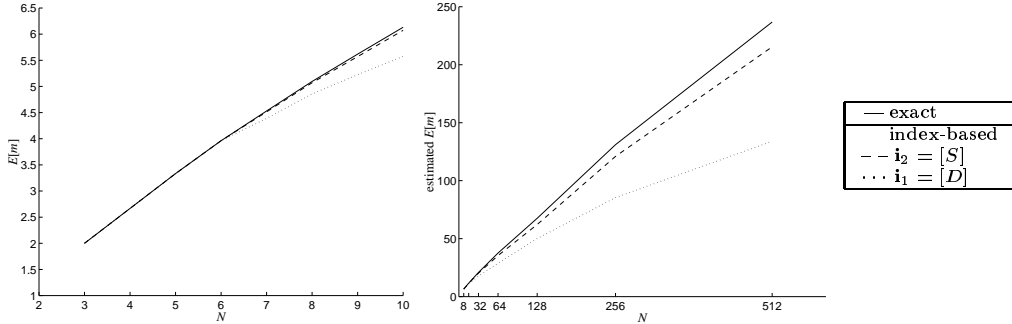


Figure 18: Number of matched vertices as a function of tree size N for complete sets up to $N = 10$ vertices (left) and for sampled sets of larger trees, 15 trees in each (right).

As the number of trees explodes as size increases, it is not practical to exhaustively generate and test large trees. Thus, another test set consisted of seven sets of fifteen nonisomorphic random trees. In each set, the trees had the same size: 8, 16, 32, 64, 128, 256 and 512 vertices. Practically, the trees were generated such that children were added to randomly chosen vertices until the desired size was obtained; if a new tree was detected isomorphic (by Algorithm 1) to an already generated tree, it was rejected. The experiment involved matching the $(15 \cdot 14)/2 = 105$ pairs within each set. The results are shown in Fig. 18 (right).

The success rate in matching, that is, the average number of matched vertices divided by the average maximum (the average number of exactly matched vertices) is shown in Fig. 19 (left). Let us consider the trade-off between matching success and computational effort. The cost of the full 100% success rate, the computational complexity of the exact `fixedTreeMatching` (Alg. 1), is N^p time units, where p is between 2 and 3 (see p. 21). The proposed heuristic version (Alg. 2) can be thought of being developed from Alg. 1 by changing the order of the commands and adding the computation of indices; one may estimate that the absolute number of additional computations, c , is around 10; it is above 1 but certainly below 100. Thus, strictly speaking, the overall computational complexity is $\mathcal{O}(cN \log \bar{D})$ (see p. 33), keeping in mind that the success rate in matching is between 91% and 99% for trees of up to $N = 512$ vertices (Fig. 18).

In Fig. 19, some curves of computation effort per success rate are displayed. The averages of actual values of \bar{D} of the test set (Fig. 18, right) were applied in computing the curves for heuristic matching. Fig. 19 can be interpreted for instance as follows: if the computational complexity of the exact `weightedBipartiteMatching` is beyond $\mathcal{O}(n^{2.5})$, and if the matching accuracy ($\geq 91\%$) of the heuristic `fixedTreeMatching` is accepted, it is better to use the heuristic algorithm, practically, for trees no matter how large.

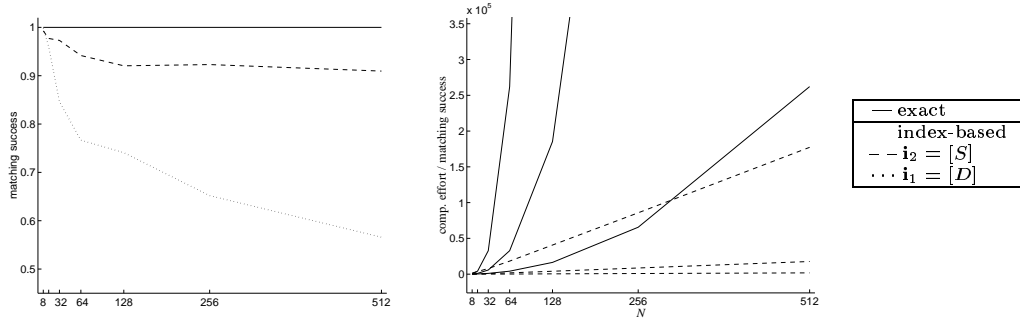


Figure 19: Success rate of index-based matching (left) and absolute computational effort per success rate in exact matching and index-based matching applying $\underline{i}_2 = [S]$ (right), assuming N^2 , $N^{2.5}$, or N^3 operations for exact matching and $N \log_2 \overline{D}$, $10N \log_2 \overline{D}$, or $100N \log_2 \overline{D}$ for index-based matching.

Conclusions. There are some interesting general observations considering the overall performance of index-based matching. To start with, using any one of the indices improves matching performance remarkably from that obtained by random matching (Fig. 17). The performance of the complete index set does not exceed that of the best index pair, and several combinations of three or four indices remain inferior to single indices or pairs of indices. Thus, it seems sufficient to use only one or two indices at least when *maximal matching* is applied in `weightedBipartiteMatching`. The tendency of getting worse results when using several indices (features) is a general observation in pattern recognition (Devijver and Kittler 1982, Bishop 1995). One of the reasons is in the fact that the information carried by indices is often conditional. For example, standard deviation of child count is distinguishing only in the case where the trees-to-be-matched are already known to be at least coarsely similar.

The performance of index-based matching in isomorphic matching is attractive, but it must be remembered that matching of isomorphic trees is an easier problem than general tree matching. Also, recalling the discussion on enumeration of trees, it is possible to design “cryptic” indices which map trees to diverse values — which is ideal in isomorphic matching — but *without being able to provide any information about the degree of similarity of two nearly isomorphic trees*. Thus, success in isomorphic matching is not a sufficient but a necessary condition for an index (combination) to be taken into account. The role of such “cryptic” indices is to complete the information provided by the structurally more guaranteeing indices.

The irregular pattern of the matching success of different indices in Fig. 17 suggests that the performance of the indices is generally sensitive to applied data. Here, the topological indices were used in the form they are defined in Table 4, p. 30: for simplicity and generality, no application-dependent scaling of indices or other optimization was applied. For the same reasons, matching performance of attribute trees, and hence, attribute indices (Table 5 on p. 31), were not discussed either.

7 Computer vision applications

In the previous sections, trees have been discussed at a general level without references to applications. This section discusses current applications and outlines possible future extensions. First, extraction of trees and attribute trees is presented as a technical prerequisite for the applications discussed.

7.1 Spatial hierarchy and trees

Technically, a digital image is a two-dimensional array of gray levels, or intensities, referring to frequencies beyond the visible range. More generally, almost anything that can be tractably stored in a two-dimensional array can be interpreted as an image: ranges, temperatures, agricultural productivities, or densities of population, for example. In many applications, images are processed one pixel at a time, assuming that groups of pixels possess no complementary information; the image array serves only as a medium for displaying input and output. In the field of computer vision, the scope generally extends from distinct pixels to higher levels of perception — from image primitives (points, line segments, edges) to more complex aggregations, and ultimately to semantic descriptions.

Texture is two-dimensional structure of repeated patterns; repetition and patterns are defined in statistical and/or syntactical terms. It has been suggested that intensity and texture should be seen as a dual nature of an image: images with pronounced, repeated intensity patterns have textural nature whereas in other images, the intensity is a dominant property (Haralick et al. 1973, Haralick 1979). A taxonomy of textured and non-textured images, together with a practical algorithm for detecting texture, is proposed by Karu et al. (1996).

Segments are areas of consistent texture, intensity and/or other property. Appropriate aggregations of elements and segments form objects. Like between intensity and textures, the distinction between objects and intensity as well as between objects and texture is vague. Clouds are an excellent example of objects mixing all these properties.

Spatial relations between objects can be presented and analysed by means of graphs. Registering all the mutual relations between objects leads to complete graphs — and intractable computational efforts as discussed in Sec. 2.3. Therefore, one should somehow limit the number of relations, that is, the number of graph edges. In the other extremity, some images can be seen as a flat mosaic of segments such that all the adjacency relations can be neglected.

If an image contains an object *hierarchy*, a tree is the most tractable structure for presentation. For example, any image that can be interpreted analogously to curves of terrain elevation on a topographic map can also be transformed to a tree as illustrated in Fig. 20. More generally, the segments are not required to have an ordered property, like altitude of elevation curves; *adjacent* and *inside* are sufficient relations for determining

a topology. Practically, the basic scheme is to recursively detect nesting image segments and construct a tree presenting the topology of segments. A scheme, called TRIM for *Tree Image*, is suggested with further detail in Publ. II and Publ. VII. How much information is preserved, and how compactly a tree captures the information, depends on the application.

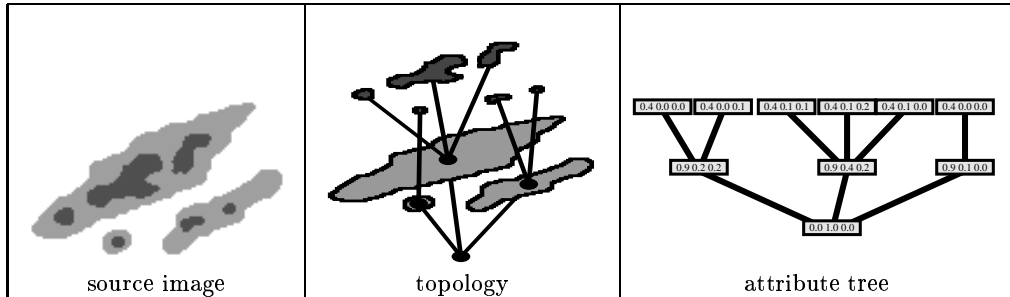


Figure 20: Extracting tree topologies in image segments.

One can also obtain trees by skeletonizing segments by means of morphological operations (Haralick et al. 1987). Recently, skeletonizing has been applied by Zhu and Yuille (FORMS, 1996) and Liu and Geiger (1999) for animate objects and by Pelillo et al. (1999b) for visual shapes. Borgefors et al. (1997,1999) have introduced a recursive, hierarchical skeletonizing scheme: again, the hierarchy means that matching can be carried out in a top-down fashion, starting from the most salient features first.

7.2 Attributes

The vertices of a tree can be attached attributes (Fig. 20, right) characterizing the referred components of objects, segments. The attributes compactly present properties of such segments. Attribute can communicate some global property, say the intensity or the average over a textural feature of the segment, or a distinct observation obtained somewhere inside the segment. It must be noted that segment properties, like intensities, obeying some partial order do not strictly follow the vertical order of the corresponding vertices in a tree. For example, in all-sky images, the intensity of the sky and holes in clouds are the same but topologically clouds are between the sky and holes (holes are inside clouds and clouds are inside the sky).

A segment can be also characterized by *shape descriptors* such as those based on moments of segments pixels (Hu 1962, Gupta and Srinath 1987). Recently, Iivarinen and Visa (1996), Iivarinen et al. (1997) and Peura and Iivarinen (1997) have studied descriptors that are based on contours and have the convenient linear-time complexity, that is, $\mathcal{O}(N)$ operations for a contour consisting of N pixels. As topological indices (Sec. 4.2) can be used for avoiding exhaustive matching of tree topologies, shape indices can be seen as a means for avoiding matching of shapes.

In some applications it may be useful to actually match the shapes — optimally or suboptimally. Mokhtarian et al. (1996) have applied *scale-space* approach for detecting dominant points of a curve. In matching two shapes, it is assumed that matching the dominant points solves the global matching problem with reasonable accuracy, and intermediate points can be matched directly by linear fitting. If shape matching is a subproblem in tree matching, the applied algorithms ought to be quick. Heuristics can be used in the detection of dominant points (Fdez-Valdivia et al. 1995). For example in cloud tracking discussed in the next section, it would be useful not only to interpolate tree topologies but also the segment contours in subsequent images.

At first sight, attribute trees extracted as in Fig. 20 seem to associate shape information with topology in somewhat coarse manner. However, there are properties which lie in between topology and shapes, making their distinction vague. Beyond the basic spatial relations *adjacent* and *inside*, Hsu et al. (1996) presents a taxonomy of relations like *bulging-into*, *slightly-touching* and *nearby-but-not-surrounding*. Nabil et al. (1996) prefer using projections in extracting similar relations with emphasis on directional order and overlappings.

7.3 Weather radar images

A glimpse in current meteorological literature reveals that most of the related research is directly associated with physical models. Basic pattern recognition methods are regarded interesting and applicable in some meteorological problems (Pankiewicz 1995), but as a rule, physical models are preferred. Meteorological models typically consist of massive arrays of physical quantities; the arrays are initialized with numerical measurements acquired by means of atmospheric soundings and land-based observations. In the models, the basic calculations are themselves simple; the underlying power is the fidelity to physical equations, high spatial resolution and massive iterations.

Among meteorologists, it is agreed that cloud observations provide significant information for weather analysis. Cloudiness and cloud genera are one of the standard weather observations of the world-wide observation network managed by the World Meteorological Organization (WMO 1956). In physical models, however, it has proven to be difficult to utilize cloud observations — be they acquired by satellites, radars, land-based detectors, or human observers. The reason is the absence of a practical mathematical formulation of the relation between cloud observations and appropriate physical quantities. Nevertheless, sky conditions are used as a valuable detail verifying or indicating the existence, approach, or development of weather phenomena.

Since 1991, automated cloud classification has been developed at the Laboratory of Computer and Information Science at Helsinki University of Technology. The first project, carried out in cooperation with the Finnish Meteorological Institute, focused on NOAA-AVHRR satellite image processing, applying local textural and spectral features (Iivarinen et al. 1995a, Iivarinen et al. 1995b, Visa and Iivarinen 1997). The second project,

focusing on land-based cloud classification was initiated by an industrial company, Vaisala Oyj, in 1995. The scope of analysis extended from local image processing towards object-oriented perception of clouds (Peura et al. 1996). The third project, *analysis of weather radar images* has continued the series of studies on cloud classification. The objective, as formulated together with the Finnish Meteorological Institute in 1996, is to develop pattern recognition methods for radar-based rain analysis and forecasting, especially short-time forecasting, *now-casting*.

Weather radars are designed to detect *precipitation*: drizzle, hail, rain, snow, etc. The typical transmission frequency is between 3 GHz and 10 GHz, compromising between the range of penetration and detection of small droplets. Radars measure the intensity, distance and direction of reflected radiation; Doppler radars measure also radial velocity. It should be stressed that even a weather radar does not detect direct physical quantities like density of water in air: basically, a pixel (voxel) in radar data corresponds to a nonlinear volume integral that is proportional to the *sum of sixth powers of droplet radii* (Rinehart 1991). That is, water content cannot be derived unambiguously but can be approximated by assuming certain distributions. Further, the contribution of water particles depends on their phase — liquid, solid, or mixed. Despite these limitations, a radar provides essential information for meteorologists. Above all, a radar image is a real-time estimate of precipitation with high spatial resolution. Also short-time predictions of rain are obtained by viewing a series of latest images and/or by applying Doppler velocities. Secondly, radar images verify and specify the development of weather conditions predicted by a coarser means such as a physical model. Also beyond operational meteorology, weather radars have promoted the understanding of the life cycles of precipitative clouds and thunderstorms.

As no physical quantities are captured by radars, meteorologists apply so called *radar parameters*, which can be thought as virtual quantities. The basic radar parameters are reflectance (dBz), Doppler velocity, and polarization. Maximum reflectance, maximum altitude of echo, and vertical integral of reflectance are examples of derived parameters. Research in radar meteorology has yielded a series of equations estimating weather from radar parameters.

Evidently, the absence of direct physical models for radar data and, hence, the artificial nature of radar parameters, in contrast with the large field of potential applications in view, have promoted using artificial intelligence in radar meteorology. Using the terminology of pattern recognition, input is assumed to have a consistent relation with target output, but the relation is vague and thus suited to processing by some artificial scheme. However, due to reliability and visibility, radar meteorologists have preferred simple techniques to more sophisticated methods. Many state-of-art approaches are based on as simple techniques as thresholding, applied for instance by Listemaa and Biggerstaff (1997) for cloud type classification and by Moszkowicz et al. (1994) for detecting anomalous propagation; Moszkowicz (1994) has also applied more general linear discriminant functions. Neural networks have been used for radar-based forecasting (Shinozawa et al.

1994, Ochiai et al. 1995), hurricane tracking (Johnson and Lin 1995), and clutter detection (da Silveira and Holt 1997). Cluttered images have been restored also by Markov chains (Upton and Fernández-Durán 1998). Larsen (1995) used Gaussian kernels for tracking and extrapolating precipitation cells; the method employed Kalman filtering for detecting size variations, translation, rotation, and appearance-disappearance of kernels in subsequent images. In a similar forecasting task, Otsuka et al. (1999) recently proposed extracting features from a down-sampled radar image for retrieving similar image sequences from a database and constructing predictions based on the retrieved sequences.

Weather radar images contain information in various scales — from clouds of constant rain, hundreds of kilometres in size, to quickly developing, local thunderstorms and showers within the scale of a kilometre. Extraction of such information requires high-level image analysis dynamically adapting to varying form and complexity of images. Consequently, there is the risk that occasionally complex samples may stall the flow of processing. Hence, the involved algorithms should have at most, say, quadratic time complexity, let alone exponential complexity. Radar images are obtained typically in every five or ten minutes, which also determines the optimal time span for related applications, and therefore, processing.

The tree extraction and matching based methods presented in this thesis seem especially feasible for the analysis of weather radar images. Fig. 21 gives an impression of the power of trees in capturing compactly the visual information of radar images. Such trees, for example, could be applied in the weather map presented in Fig. 12. Further demonstrations of applications — monitoring, image retrieval, learning — are presented in Publications IV – VII.

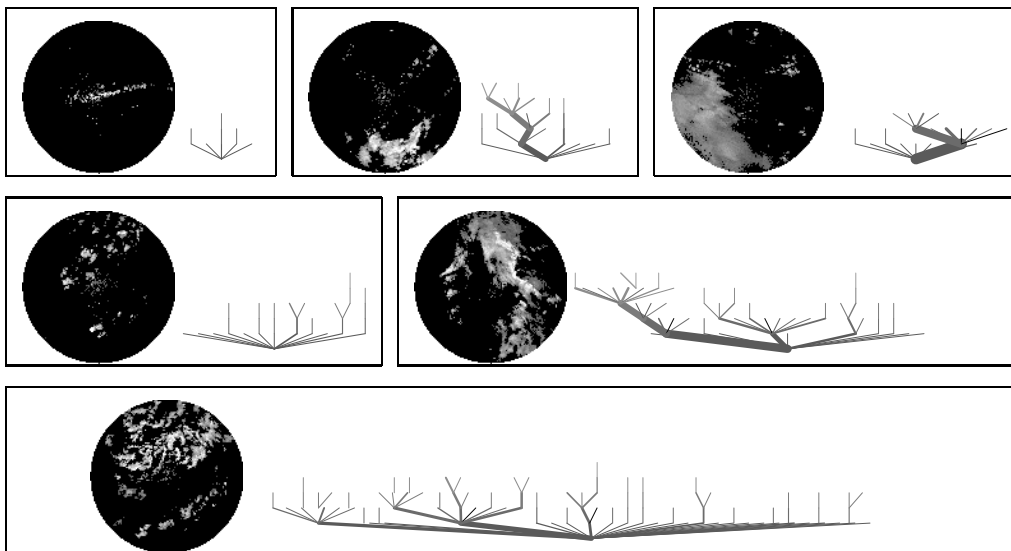


Figure 21: Weather radar images and corresponding trees. The intensity of a segment appears as the brightness of the corresponding edge. The weight (visualized as edge thickness) has been defined as an increasing function of segment size and intensity.

When matching unordered trees, the implicit assumption is that the metric locations of the objects are irrelevant. However, one should here note that while the proposed approach is essentially based on adjacency relations (topology), also coordinates (topography) can be utilized as attributes: cloud tracking outlined in Publ. IV applies coordinates of segment centroids under the assumption that the clouds in subsequent images undergo relatively small translations, rotations and scaling.

7.4 Auroral images

Geophysical Research at the Finnish Meteorological Institute studies the dynamics of the Earth's magnetosphere. Northern light (*Aurora Borealis*) is an indication of magnetospheric activity, thus auroral images are captured by several all-sky cameras located in Northern Scandinavia and Finland (Syrjäsuo et al. 1998). A sample image is shown in Fig. 22. Naturally, detection of activity is the primary interest, but another goal is to distinguish between different types of aurora in order to detect correlations between the images and electromagnetic disturbances in human activities as well as aiming at better understanding of the phenomenon itself. The current approach, suggested by Syrjäsuo and Pulkkinen (1999), is based on skeletonizing (Fig. 22 b).

While a skeleton is a description of the outline of a gray-level slice or the average outline of several slices, the approach proposed in this thesis (Sec. 7.2) produces a tree which preserves information of all the gray levels of an image (Fig. 22 c). In addition, attributes (Sec. 7.2) can be readily attached in segment topology approach, although one could develop attributes for skeletal trees as well. Publ. IV and Publ. VII demonstrate monitoring and distance calculation of auroral images.

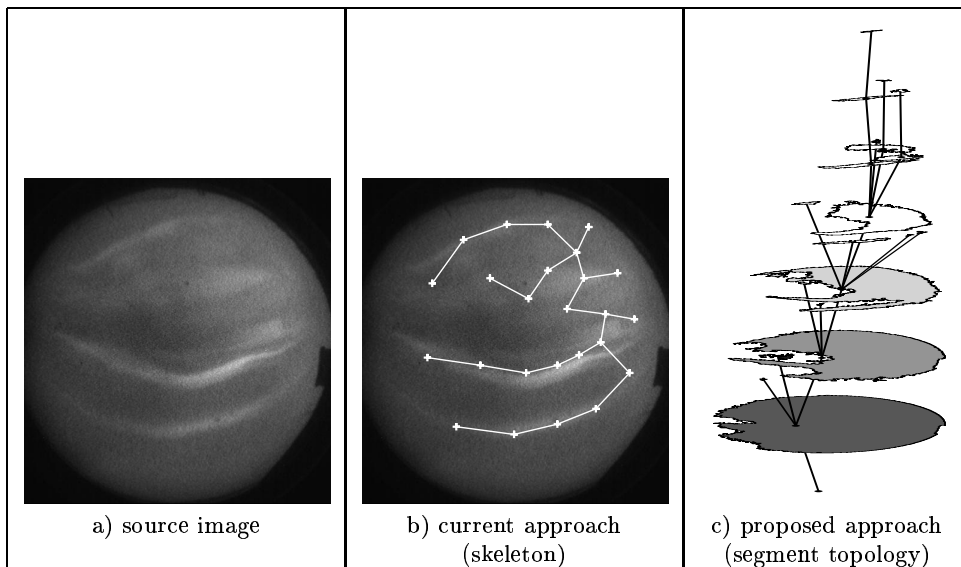


Figure 22: Two tree-based approaches for extracting information from auroral images.

7.5 Known problems

Within pattern recognition, an object model is considered feasible if the similarity of two objects can be reliably assessed by deriving the similarity between the extracted models. In extracting segment topologies as suggested in Fig. 20, an interesting question is how well an obtained tree corresponds to the original image. One of the encountered problems is demonstrated in Fig. 23: Visually small variations of weakly connected segments may cause topological discontinuities due to pixel-wise sensitivity of crisp adjacencies. Circular compositions, “atolls”, are the most awkward, as concavities may suddenly become internal segments. On the other hand, in some applications the distinction may be desired. If a recognition task is considered, the brute force approach would be to collect enough samples such that different variations get covered statistically. A sophisticated way is to detect fuzzy adjacencies (Bloch and Maitre 1997). A tempting practical technique would be to simply “extrapolate” the spatial topology by spreading segments with morphological operators: the obtained artificial topology would correspond to the distances.

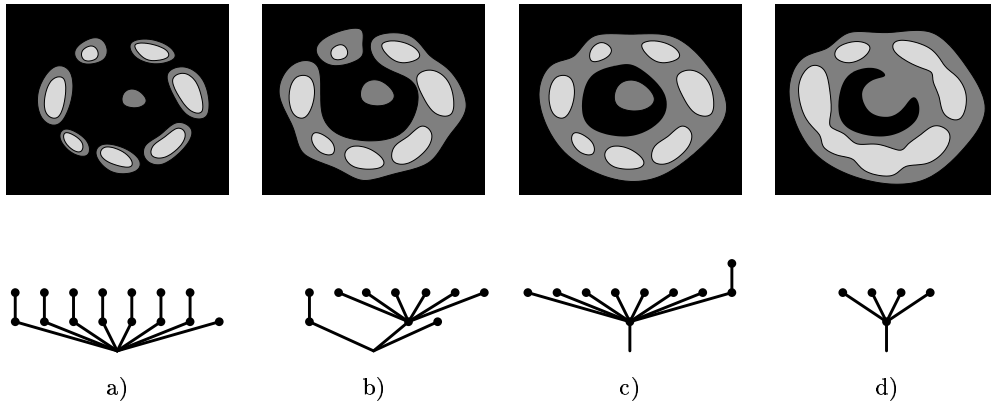


Figure 23: The “atoll problem”. Visually similar objects correspond to diverging topologies.

Tree topology does not cover spatial organization beyond the relations adjacent and inside: objects can be moved, rotated, arranged in rows etc. in the image space without changes in the perceived topology. However, the assumption that seems valid in radar and auroral images is that spatially close segments are connected by a common parent segment, thus the extracted tree corresponds fairly well to the “correct” visual impression. Issues related to practical problems topology extraction are further discussed in Publ. II as well as in (Peura 1998).

7.6 Future applications

The applications discussed so far have served as primary test benches for the developed techniques. Several directions remain still to be explored. Above all, modelling by FLEXIBLE TREE MATCHING (p. 22) and the involved heuristics should be developed. On the other hand, it is obvious that the techniques developed already are applicable in a variety of problems involving tree indexing and matching.

For example, there are many medical applications in which tree matching is involved in a form or another. For example, Pisupati et al. (1996) have used geometrically constrained tree matching for finding correspondences between subsequent 3D images of lungs. Also the *confinement trees* applied by Mattes et al. (1999; Mattes and Demongeot (1999)), for medical images seem rather similar to the trees extracted by TRIM algorithms (Fig. 20, Publ. II, Publ. VII).

In geography, graphs are natural models of human activities. Certain transportation networks, river basins and deltas are examples of tree-like objects. On the other hand, *shape descriptors* somewhat similar to those proposed in Publ. I have been applied in analysing shapes of cities (Griffith et al. 1986, Medda et al. 1998). Intuitively, it seems feasible to model urban areas as hierarchical clusters, as Fig. 24 suggests. In tree-matching based analysis, one could use some basic continuous quantities (like action or population density) for constructing topologies, and use semantic classes (land use, type of activity) as attributes.

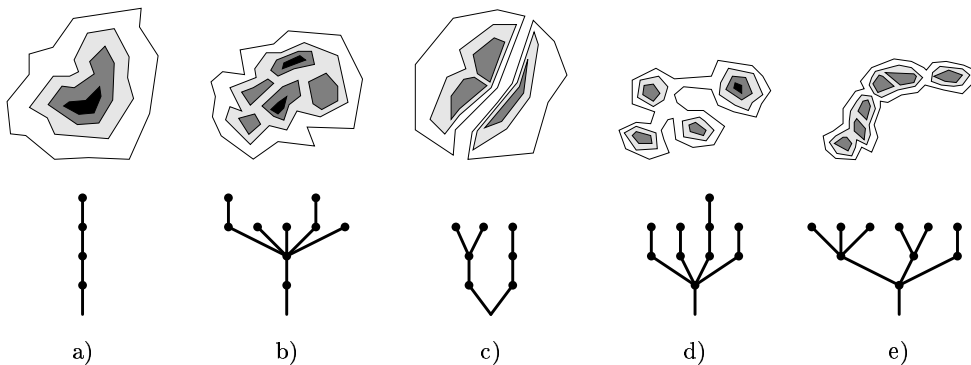


Figure 24: Topologies of hypothetical cities.

8 Conclusions

This thesis presented new techniques for applying unordered rooted attribute trees in adaptive image analysis. The emphasis was on computer vision applications, but the proposed techniques seem generally applicable in pattern recognition.

The discussion started by approaching tree matching as a graph theoretic problem. Definitions were given, and the computational complexity of tree matching and related problems was discussed. While exact formulations provide a convenient basis for theoretical analysis, pattern recognition in general involves detecting irregular and indefinite objects, which implies that inexact models and algorithms must be considered as well.

The proposed heuristic tree matching scheme, Algorithm 2, provided a practical means for solving a tree matching problem (FIXED TREE MATCHING, Def. 7) in nearly linear time, instead of the quadratic or cubic time required by the exact algorithms. The basic idea was to divide a tree into subtrees, and present each subtree as a set of indices, and to match the subtrees recursively by means of the indices. The proposed approach challenges the related state-of-art heuristics. No comparative studies were presented here but the computational simplicity and the performance of the proposed method seem attractive with respect to other reported methods.

The discrete nature of trees was attacked by applying vertex weights. While introducing the weights and other attributes provided flexibility, it also required disciplined design in order to obtain consistency over the applied operators and algorithms. The suggested continuous tree-to-tree distance metrics, matching and mixing schemes were defined such that consistency issues were taken into account. The main contribution of this thesis, the Self-Organizing Map of Attribute Trees combined the proposed techniques to a powerful adaptive system to be used for classifying and monitoring of hierarchical objects, as well as for visualizing involved distributions.

Principles for extracting hierarchical objects from images, together with an outline of an algorithm, were also presented. Analysis of weather radar images has been the pilot application for this study, and the results obtained this far are promising. Similar experiments on auroral images also yielded interesting results.

To conclude, this thesis presented various tools that can be applied in the analysis of irregular hierarchical images. The proposed techniques perform quickly, accurately and consistently. One can expect that the proposed methods are feasible to various tasks also beyond the applications presented in this thesis.

References

- Ahuja, R. K., Orlin, J. B. and Tarjan, R. E. (1989). Improved time bounds for the maximum flow problem, *SIAM Journal on Computing* **18**(5): 939–954.
- Ashbrook, A. P., Thacker, N. A. and Rockett., P. I. (1995). Pairwise geometric histograms: A scaleable solution for the recognition of 2D rigid shapes., *9th Scandinavian Conference on Image Analysis (SCIA '95)*, Vol. 1, Uppsala, Sweden, pp. 271–278.
- Avis, D. (1983). A survey of heuristics for the weighted matching problem, *Networks* **13**(4): 475–494.
- Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*, Clarendon Press, Oxford.
- Bloch, I. and Maître, H. (1997). Extending adjacency to fuzzy sets for coping with imprecise image objects, *9th International Conference on Image Analysis and Processing (ICIAP'97)*, Vol. 1, Florence, Italy, pp. 30–37.
- Bondy, J. A. and Murty, U. S. R. (1976). *Graph Theory with Applications*, MacMillan.
- Borgefors, G., Ramella, G. and Baja, G. S. d. (1997). Multiscale skeletons via permanence ranking, in C. Arcelli, L. P. Cordella and G. S. di Baja (eds), *Advances in Visual Form Analysis*, World Scientific, pp. 31–42. Proceedings of the 3th International Workshop on Visual Form (IWVF3), Capri, Italy.
- Borgefors, G., Ramella, G. and Baja, G. S. d. (1999). Permanence-based shape decomposition in binary pyramids, *10th International Conference on Image Analysis and Processing (ICIAP'99)*, IEEE Computing Society, pp. 38–43.
- Buss, S. R. and Yianilos, P. N. (1998). Linear and $O(n \log n)$ time minimum cost matching algorithms for quasi-convex tours, *SIAM Journal on Computing* **27**(1): 170–201.
- Choy, C. S.-T. and Siu, W.-C. (1995). Algorithm for solving bipartite subgraph problem with probabilistic self-organizing learning, *1995 International Conference on Acoustics, Speech, and Signal Processing. Conference Proceedings*, Vol. 5, Dept. of Electron. Eng., Hong Kong Polytech. Univ, Kowloon, Hong Kong, IEEE, New York, NY, USA, pp. 3351–4.
- da Silveira, R. B. and Holt, A. (1997). A neural network application to discriminate between clutter and precipitation using polarization information as feature space, *Proceedings of the 28th conference on radar meteorology*, American Meteorological Society, Austin, Texas, pp. 57–58.
- de Bruijn, N., Knuth, D. E. and Rice, S. (1972). The average height of planted plane trees, in R. C. Read (ed.), *Graph Theory and Computing*, Academic Press, pp. 15–22.
- Devijver, P. and Kittler, J. (1982). *Pattern recognition: A statistical approach*, Prentice-Hall International, London.

- Eshera, M. and Fu, K.-S. (1984). A graph distance measure for image analysis, *IEEE Transactions on Systems, Man, and Cybernetics* **14**(3): 398–408.
- Fdez-Valdivia, J., García, J. and de la Blanca, N. P. (1995). A new approach to 2-D shapes characterization, *9th Scandinavian Conference on Image Analysis (SCIA '95)*, pp. 663–669.
- Fu, K.-S. and Bhargava, B. K. (1973). Tree systems for syntactic pattern recognition, *IEEE Transactions on Computers* **C-22**(12): 1087–1098.
- Fujimura, K., Obu-Cann, K. and Tokutaka, H. (1999). Optimization of surface component mounting on the printed circuit board using SOM-TSP method, *International Conference on Artificial Neural Networks*, Edinburgh, Scotland, pp. 643–647.
- Garey, M. R. and Johnson, D. S. (1979). *Computers and intractability: A guide to the theory of NP-completeness*, Freeman, San Francisco, CA.
- Griffith, D. A., O'Neill, M. P., O'Neill, W. A., Leifer, L. and Mooney, R. G. (1986). Shape indices: Useful measures or red herrings, *Professional Geographer* **38**(3): 263–270.
- Gupta, L. and Srinath, M. (1987). Contour sequence moments for the classification of closed planar shapes, *Pattern Recognition* **20**(3): 262–272.
- Haralick, R. M. (1979). Statistical and structural approaches to texture, *Transactions of the IEEE* **67**(5): 786–803.
- Haralick, R. M., Shanmugam, K. and Dinstein, I. (1973). Textural features for image classification, *IEEE Transactions on Systems, Man, and Cybernetics* **SMC-6**(6): 610–621.
- Haralick, R. M., Sternberg, S. R. and Zuang, X. (1987). Image analysis using mathematical morphology, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-9**(4): 532–550.
- Herbin, S. (1995). Graph matching by self-organizing feature maps, in F. Fogelman-Soulié and P. Gallinari (eds), *Proc. ICANN'95, Int. Conf. on Artificial Neural Networks*, Vol. II, EC2, Nanterre, France, pp. 57–62.
- Hopcroft, J. E. and Karp, R. M. (1973). An $n^{\frac{5}{2}}$ algorithm for maximum matchings in bipartite graphs, *SIAM Journal on Computing* **2**(4): 225–231.
- Hsu, C.-C., Chu, W. W. and Taira, R. K. (1996). A knowledge-based approach for retrieving images by content, *IEEE Transactions on Knowledge and Data Engineering* **8**(4): 522–532.
- Hu, M.-K. (1962). Visual pattern recognition by moment invariants, *IRE Transactions on Information Theory* **8**(2): 179–187.
- Iivarinen, J. and Visa, A. (1996). Shape recognition of irregular objects, *Intelligent Robots and Computer Vision XV: Algorithms, Techniques, Active Vision, and Materials Handling, Proc. SPIE 2904* pp. 25–32.

- Iivarinen, J., Peura, M. and Visa, A. (1995a). Verification of a multichannel cloud classifier, *9th Scandinavian Conference on Image Analysis (SCIA '95)*, pp. 591–595.
- Iivarinen, J., Peura, M., Särelä, J. and Visa, A. (1997). Comparison of combined shape descriptors for irregular objects, in A. F. Clark (ed.), *The Eighth British Machine Vision Conference (BMVC'97)*, Vol. 2, pp. 430–439.
- Iivarinen, J., Valkealahti, K., Visa, A. and Simula, O. (1995b). Development of a cloud classifier, *Technical Report A25*, Helsinki University of Technology, Laboratory of Computer and Information Science.
- Imiya, A., Fujiwara, Y. and Kawashima, T. (1996). Reconstruction, recognition and representation of trees, *13th International Conference on Pattern Recognition (ICPR'96)*, Vienna, Austria, Vol. 4, IAPR, IEEE Computing Society, pp. 595–600.
- Johnson, G. P. and Lin, F. C. (1995). Hurricane tracking via backpropagation neural network, *Proceedings of the IEEE International Conference on Neural Networks*, Vol. 2, Perth, Australia, pp. 1103–1106.
- Kann, V. (1992). *On the Approximability of NP-complete Optimization Problems*, Ph.D. thesis, Royal Institute of Technology, Stockholm, Sweden.
- Karu, K., Jain, A. K. and Bolle, R. M. (1996). Is there any texture in the image?, *13th International Conference on Pattern Recognition (ICPR'96)*, Vienna, Austria, Vol. 2, IAPR, IEEE Computing Society, pp. 770–774.
- Kilpeläinen, P. (2001). Private communication.
- Kilpeläinen, P. and Mannila, H. (1994). Query primitives for tree-structured data, in M. Crochemore and D. Gusfield (eds), *the 5th Annual Symposium on Combinatorial Pattern Matching*, Springer-Verlag, pp. 213–225.
- Kilpeläinen, P. and Mannila, H. (1995). Ordered and unordered tree inclusion, *SIAM Journal on Computing* **24**(2): 340–356.
- Kohonen, T. (1982). Self-organized formation of topologically correct feature maps, *Biological Cybernetics* **43**: 59–69.
- Kohonen, T. (1985). Median strings, *Pattern Recognition Letters* **6**(3): 309–313.
- Kohonen, T. (1990). The Self-Organizing Map, *Transactions of the IEEE* **78**(9): 1464–1480.
- Kohonen, T. (1995). *Self-Organizing Maps*, Springer Series in Information Sciences, Springer, Berlin
- Kohonen, T. (1996). Self-organizing maps of symbol strings, *Technical Report A42*, Helsinki University of Technology, Laboratory of Computer and Information Science.
- Kohonen, T. and Somervuo, P. (1998). Self-organizing maps of symbol strings, *Neurocomputing* **21**: 19–30.

- Kohonen, T., Oja, E., Simula, O., Visa, A. and Kangas, J. (1996). Engineering applications of the self-organizing map, *Transactions of the IEEE* **84**(10): 1358–1384.
- Koikkalainen, P. (1994). Progress with the tree-structured self-organizing map, in A. G. Cohn (ed.), *Proc. ECAI'94, 11th European Conf. on Artificial Intelligence*, John Wiley & Sons, New York, pp. 211–215.
- Kreher, D. L. and Stinson, D. R. (1999). *Combinatorial Algorithms — Generation, Enumeration and Search*, CRC Press.
- Larsen, M. (1995). A Kalman filter for tracking rain cells, *9th Scandinavian Conference on Image Analysis (SCIA '95)*, pp. 679–688. Uppsala, Sweden.
- Lawler, E. L. (1976). *Combinatorial Optimization — Networks and Matroids*, Holt, Rinehart and Winston.
- Levenshtein, V. I. (1965). Binary codes capable of correcting spurious insertions and deletions of ones (original in russian), *Russian Problemy Peredachi Informatsii* **1**(1): 12–25.
- Listemaa, S. and Biggerstaff, M. I. (1997). An improved scheme for convective/stratiform echo classification using radar reflectivity, *Proceedings of the 28th conference on radar meteorology*, American Meteorological Society, Austin, Texas, pp. 274–275.
- Liu, T.-L. and Geiger, D. (1999). Approximate tree matching and shape similarity, *7th International Conference on Computer Vision (ICCV'99)*, Kerkyra, Greece, IEEE, pp. 456–462.
- Mannila, H. (2000). Private communication.
- Mattes, J. and Demongeot, J. (1999). Tree representation and implicit tree matching for a coarse to fine image matching algorithm, in C. Taylor and A. Clochester (eds), *Medical Image Computing & Computer-Assisted Intervention (MICCAI'99) — Second International Conference*, Vol. 1679 of *Lecture Notes in Computer Science*, Springer, Cambridge, England, pp. 646–655.
- Mattes, J., Richard, M. and Demongeot, J. (1999). Tree representation for image matching and object recognition, in G. Bertrand, M. Couprie and L. Perrotton (eds), *Discrete Geometry for Computer Imagery — 8th International Conference (DGCI'99)*, Vol. 1568 of *Lecture Notes in Computer Science*, Springer, Marne-la-Vallée, France, pp. 298–309.
- Matula, D. W. (1968). An algorithm for subtree identification, *SIAM Review* **10**(2): 260–261.
- Medda, F., Nijkamp, P. and Rietveld, P. (1998). Recognition and classification of urban shapes, *Geographical Analysis* **30**(3): 305–314.

- Mokhtarian, F., Abbasi, S. and Kittler, J. (1996). Robust and efficient shape indexing through curvature scale space, *The Seventh British Machine Vision Conference (BMVC'96)*, Edinburgh, UK, pp. 53–62.
- Moszkowicz, S. (1994). Algorithms for meteorological phenomena recognition and AP echoes suppression on automated weather radar system AMSR, *Proceedings of COST-75: Weather radar systems*, European Commission, Brussels, Belgium.
- Moszkowicz, S., Ciach, G. J. and Krajewski, W. F. (1994). Statistical detection of anomalous propagation in radar reflectivity patterns, *Journal of Atmospheric and Oceanic Technology* **11**: 1026–1034.
- Motwani, R. and Raghavan, P. (1997). *Randomized Algorithms*, Cambridge University Press.
- Nabil, M., Ngu, A. H. H. and Shepherd, J. (1996). Picture similarity retrieval using the 2D project interval representation, *IEEE Transactions on Knowledge and Data Engineering* **8**(4): 533–539.
- Ochiai, K., Suzuki, H., Shinozawa, K., Fujii, M. and Sonehara, N. (1995). Snowfall and rainfall forecasting from weather radar images with artificial neural networks, *Proceedings of the IEEE International Conference on Neural Networks*, Vol. 2, Perth, Australia, pp. 1182–1187.
- Otsuka, K., Horikoshi, T., Suzuki, S. and Kojima, H. (1999). Memory-based forecasting of complex natural patterns by retrieving similar image sequences, *10th International Conference on Image Analysis and Processing (ICIAP'99)*, Venice, Italy, IAPR, IEEE Computing Society, pp. 874–879.
- Pankiewicz, G. S. (1995). Pattern recognition techniques for the identification of cloud and cloud systems, *Meteorological Applications* **2**(2): 257–271.
- Papadimitriou, C. (1994). *Computational complexity*, Addison-Wesley.
- Pearl, J. (1984). *Heuristics: intelligent search strategies for computer problem solving*, Addison-Wesley.
- Pelillo, M., Siddiqi, K. and Zucker, S. W. (1998). Matching hierarchical structures using association graphs, *Proceedings of ECCV'98, 5th European Conference on Computer Vision*, Springer Verlag, Berlin.
- Pelillo, M., Siddiqi, K. and Zucker, S. W. (1999a). Attributed tree matching and maximum weight cliques, *10th International Conference on Image Analysis and Processing, (ICIAP'99)*, IEEE Computer Society Press, Venice, Italy.
- Pelillo, M., Siddiqi, K. and Zucker, S. W. (1999b). Matching hierarchical structures using association graphs, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **21**(11): 1105–1120.

- Pelillo, M., Siddiqi, K. and Zucker, S. W. (2000). Continuous-based heuristics for graph and tree isomorphisms, with application to computer vision, in P. M. Pardalos (ed.), *Approximation and Complexity in Numerical Optimization: Continuous and Discrete Problems*, Kluwer Academic Publishers, Boston, MA.
- Peura, M. (1998). Analysis of irregular and hierarchical visual objects, Helsinki University of Technology, Laboratory of Computer and Information Science. Licentiate's thesis.
- Peura, M. and Iivarinen, J. (1997). Efficiency of simple shape descriptors, in C. A. Carlo, L. P. Cordella and G. S. di Baja (eds), *Advances in Visual Form Analysis – Proceedings of the Third International Workshop on Visual Form (IWVF3)*, World Scientific, pp. 443–451.
- Peura, M., Koistinen, J. and King, R. (1998). Visual modelling of radar images, *COST-75: Advanced weather radar systems — international seminar*, European Commission, pp. 307–317.
- Peura, M., Visa, A. and Kostamo, P. (1996). A new approach to land-based cloud classification, *13th International Conference on Pattern Recognition (ICPR'96)*, Vienna, Austria, Vol. 4, IAPR, IEEE Computing Society, pp. 143–147.
- Pisupati, C., Wolff, L., Mitzner, W. and Zerhouni, E. (1996). Geometric tree matching with applications to 3D lung structures, *Proceedings of the Twelfth Annual Symposium On Computational Geometry*, ACM Press, Philadelphia, PA, U.S.A., pp. C19–C20.
- Read, R. C. (1972). The coding of various kinds of unlabeled trees, in R. C. Read (ed.), *Graph Theory and Computing*, Academic Press, pp. 153–182.
- Reyner, S. W. (1977). An analysis for a good algorithm for the subtree problem, *SIAM Journal on Computing* **6**(4): 730–732.
- Rinehart, R. E. (1991). *Radar for Meteorologists*, 2nd edn, Department of Atmospheric Sciences, University of North Dakota, USA.
- Santini, S. and Jain, R. (1999). Similarity measures, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **21**(9): 871–883.
- Shasha, D., Wang, J. T.-L., Zhang, K. and Shih, F. Y. (1994). Exact and approximate algorithms for unordered tree matching, *IEEE Transactions on Systems, Man, and Cybernetics* **24**(4): 668–678.
- Shinozawa, K., Fujii, M., and Sonehara, N. (1994). A weather radar image prediction method in local parallel computation, *Proceedings of the IEEE International Conference on Neural Networks (ICNN'94)*, Orlando, Florida, pp. 4210–4215.

- Shokoufandeh, A., Dickinson, S., Siddiqi, K. and Zucker, S. (1999). Indexing using a spectral encoding of topological structure, *1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 1999)*, Vol. 2, IEEE Computing Society, Fort Collins, Colorado, USA, pp. 491–497.
- Somervuo, P. and Kohonen, T. (2000). Clustering and visualization of large protein sequence databases by means of an extension of the self-organizing map, *Technical Report A58*, Helsinki University of Technology, Laboratory of Computer and Information Science.
- Syrjäsuo, M. and Pulkkinen, T. (1999). Determining the skeletons of the aurora, *10th International Conference on Image Analysis and Processing (ICIAP'99)*, IAPR, IEEE Computer Society, pp. 1063–1066. Venice, Italy.
- Syrjäsuo, M. T., Pulkkinen, T., Janhunen, P., Viljanen, A., Pellinen, R., Kauristie, K., Opgenoorth, H., Wallman, S., Eglitis, P., Karlsson, P., Amm, O., Nielsen, E. and Thomas, C. (1998). Observations of substorm electrodynamics by using the miracle network, *Proceedings of the Fourth International Conference on Substorms*, pp. 111–120.
- Ullmann, J. (1976). An algorithm for subgraph isomorphism, *Journal of the Association for Computing Machinery* **23**(1): 31–42.
- Upton, G. and Fernández-Durán, J.-J. (1998). Statistical techniques for clutter removal and attenuation detection in radar reflection images, *Proceedings of COST-75: Advanced weather radar systems, Locarno, Switzerland*, European Commission.
- Vaidya, P. (1989). Geometry helps in matching, *SIAM Journal on Computing* **18**(6): 1201–1225.
- Verma, R. M. and Reyner, S. W. (1989). An analysis for a good algorithm for the subtree problem, corrected, *SIAM Journal on Computing* **18**(5): 906–908.
- Visa, A. and Iivarinen, J. (1997). Evolution and evaluation of a trainable cloud classifier, *IEEE Transactions on Geoscience and Remote Sensing* **35**(5): 1307–1315.
- WMO (1956). *International Cloud Atlas*.
- Zhang, K., Statman, R. and Shasha, D. (1992). On the editing distance between unordered labeled trees, *Information Processing Letters* **42**(3): 133–139.
- Zhu, S. C. and Yuille, A. L. (1996). FORMS: A flexible object recognition and modeling system, *International Journal of Computer Vision* **20**(3): 187–212.

A Probabilities and expectations related to coinflip trees

A.1 Expectation minimum child count

Let us consider 1-to-1 matching of two trees which have been generated by the coinflip model (Def. 15) with limited child count \bar{D} and maximum height \bar{H} . At any step of tree matching, the children of two nodes are getting matched. Assuming that D_1 and D_2 are the respective numbers of children, in 1-to-1 matching $\min(D_1, D_2)$ become matched. The expectation of $\min(D_1, D_2)$ is derived below, and is used in (34).

$$\begin{aligned}
E_{r\bar{D}} &= \sum_{i=1}^{\bar{D}} \sum_{j=1}^{\bar{D}} P^{(i)P^{(j)}} \min(i, j) \\
&= \sum_{i=1}^{\bar{D}-1} \sum_{j=1}^{\bar{D}-1} (1/2)^{i+1} (1/2)^{j+1} \min(i, j) \\
&\quad + \sum_{i=1}^{\bar{D}-1} (1/2)^{i+1} (1/2)^{\bar{D}} i + \sum_{j=1}^{\bar{D}-1} (1/2)^{\bar{D}} (1/2)^{j+1} j + (1/2)^{\bar{D}} (1/2)^{\bar{D}} \bar{D} \\
&= \frac{1}{4} \sum_{i=1}^{\bar{D}-1} \sum_{j=1}^{\bar{D}-1} (1/2)^i (1/2)^j \min(i, j) + 2 \left(\frac{1}{2}\right)^{\bar{D}} \frac{1}{2} \sum_{i=1}^{\bar{D}-1} \left(\frac{1}{2}\right)^i i + \bar{D} \left(\frac{1}{4}\right)^{\bar{D}} \\
&= \frac{1}{4} \sum_{i=1}^{\bar{D}-1} \sum_{j=1}^{i-1} \left(\frac{1}{2}\right)^{i+j} \min(i, j) + \frac{1}{4} \sum_{i=1}^{\bar{D}-1} \left(\frac{1}{2}\right)^{2i} i + \frac{1}{4} \sum_{j=1}^{\bar{D}-1} \sum_{i=1}^{j-1} \left(\frac{1}{2}\right)^{i+j} \min(i, j) \\
&\quad + \left(\frac{1}{2}\right)^{\bar{D}} \sum_{i=1}^{\bar{D}-1} \left(\frac{1}{2}\right)^i i + \bar{D} \left(\frac{1}{4}\right)^{\bar{D}} \\
&= \frac{1}{4} \sum_{i=1}^{\bar{D}-1} \sum_{j=1}^{i-1} \left(\frac{1}{2}\right)^{i+j} j + \frac{1}{4} \sum_{i=1}^{\bar{D}-1} \left(\frac{1}{4}\right)^i i + \frac{1}{4} \sum_{j=1}^{\bar{D}-1} \sum_{i=1}^{j-1} \left(\frac{1}{2}\right)^{i+j} i \\
&\quad + \frac{1}{2\bar{D}} \sum_{i=1}^{\bar{D}-1} \left(\frac{1}{2}\right)^i i + \bar{D} \left(\frac{1}{4}\right)^{\bar{D}} \\
&= \frac{1}{2} \sum_{i=1}^{\bar{D}-1} \sum_{j=1}^{i-1} \left(\frac{1}{2}\right)^{i+j} j + \frac{1}{4} \sum_{i=1}^{\bar{D}-1} \left(\frac{1}{4}\right)^i i + \frac{1}{2\bar{D}} \sum_{i=1}^{\bar{D}-1} \left(\frac{1}{2}\right)^i i + \bar{D} \left(\frac{1}{4}\right)^{\bar{D}} \\
&= \frac{1}{2} \sum_{i=1}^{\bar{D}-1} \left(\frac{1}{2}\right)^i \sum_{j=1}^{i-1} \left(\frac{1}{2}\right)^j j + \frac{1}{4} \sum_{i=1}^{\bar{D}-1} \left(\frac{1}{4}\right)^i i + \frac{1}{2\bar{D}} \sum_{i=1}^{\bar{D}-1} \left(\frac{1}{2}\right)^i i + \bar{D} \left(\frac{1}{4}\right)^{\bar{D}} \\
&= \frac{1}{2} \sum_{i=1}^{\bar{D}-1} \left(\frac{1}{2}\right)^i \left[\left(\frac{1}{2}\right) \frac{1 - (1/2)^{i-1}}{(1 - 1/2)^2} - \frac{(i-1)(1/2)^i}{1 - 1/2} \right] + \frac{1}{4} \sum_{i=1}^{\bar{D}-1} \left(\frac{1}{4}\right)^i i + \frac{1}{2\bar{D}} \sum_{i=1}^{\bar{D}-1} \left(\frac{1}{2}\right)^i i + \bar{D} \left(\frac{1}{4}\right)^{\bar{D}}
\end{aligned}$$

$$\begin{aligned}
&= \frac{1}{2} \sum_{i=1}^{\overline{D}-1} \left(\frac{1}{2}\right)^i 2 \left[1 - \left(\frac{1}{2}\right)^{i-1} - (i-1) \left(\frac{1}{2}\right)^i \right] + \frac{1}{4} \sum_{i=1}^{\overline{D}-1} \left(\frac{1}{4}\right)^i i + \frac{1}{2\overline{D}} \sum_{i=1}^{\overline{D}-1} \left(\frac{1}{2}\right)^i i + \overline{D} \left(\frac{1}{4}\right)^{\overline{D}} \\
&= \sum_{i=1}^{\overline{D}-1} \left(\frac{1}{2}\right)^i \left[1 - \left(\frac{1}{2}\right)^i - i \left(\frac{1}{2}\right)^i \right] + \frac{1}{4} \sum_{i=1}^{\overline{D}-1} \left(\frac{1}{4}\right)^i i + \frac{1}{2\overline{D}} \sum_{i=1}^{\overline{D}-1} \left(\frac{1}{2}\right)^i i + \overline{D} \left(\frac{1}{4}\right)^{\overline{D}} \\
&= \sum_{i=1}^{\overline{D}-1} \left(\frac{1}{2}\right)^i - \sum_{i=1}^{\overline{D}-1} \left(\frac{1}{2}\right)^{2i} - \sum_{i=1}^{\overline{D}-1} i \left(\frac{1}{2}\right)^{2i} + \frac{1}{4} \sum_{i=1}^{\overline{D}-1} \left(\frac{1}{4}\right)^i i + \frac{1}{2\overline{D}} \sum_{i=1}^{\overline{D}-1} \left(\frac{1}{2}\right)^i i + \overline{D} \left(\frac{1}{4}\right)^{\overline{D}} \\
&= \sum_{i=1}^{\overline{D}-1} \left(\frac{1}{2}\right)^i - \sum_{i=1}^{\overline{D}-1} \left(\frac{1}{4}\right)^i - \sum_{i=1}^{\overline{D}-1} i \left(\frac{1}{4}\right)^i + \frac{1}{4} \sum_{i=1}^{\overline{D}-1} \left(\frac{1}{4}\right)^i i + \frac{1}{2\overline{D}} \sum_{i=1}^{\overline{D}-1} \left(\frac{1}{2}\right)^i i + \overline{D} \left(\frac{1}{4}\right)^{\overline{D}} \\
&= \sum_{i=1}^{\overline{D}-1} \left(\frac{1}{2}\right)^i - \sum_{i=1}^{\overline{D}-1} \left(\frac{1}{4}\right)^i - \frac{3}{4} \sum_{i=1}^{\overline{D}-1} \left(\frac{1}{4}\right)^i i + \frac{1}{2\overline{D}} \sum_{i=1}^{\overline{D}-1} \left(\frac{1}{2}\right)^i i + \overline{D} \left(\frac{1}{4}\right)^{\overline{D}} \\
&= \left(\frac{1}{2}\right) \frac{1 - (1/2)^{\overline{D}-1}}{1 - 1/2} - \left(\frac{1}{4}\right) \frac{1 - (1/4)^{\overline{D}-1}}{1 - 1/4} - \frac{3}{4} \left[\left(\frac{1}{4}\right) \frac{1 - (1/4)^{\overline{D}-1}}{(1 - 1/4)^2} - \frac{(\overline{D}-1)(1/4)^{\overline{D}}}{1 - 1/4} \right] \\
&\quad + \frac{1}{2\overline{D}} \left[\left(\frac{1}{2}\right) \frac{1 - (1/2)^{\overline{D}-1}}{(1 - 1/2)^2} - \frac{(\overline{D}-1)(1/2)^{\overline{D}}}{1 - 1/2} \right] + \overline{D} \left(\frac{1}{4}\right)^{\overline{D}} \\
&= 1 - \left(\frac{1}{2}\right)^{\overline{D}-1} - \frac{1 - (1/4)^{\overline{D}-1}}{3} - \frac{3}{4} \left[\frac{1 - (1/4)^{\overline{D}-1}}{9/4} - \frac{(\overline{D}-1)(1/4)^{\overline{D}}}{3/4} \right] \\
&\quad + \frac{2}{2\overline{D}} \left[1 - (1/2)^{\overline{D}-1} + (1/2)^{\overline{D}} - \overline{D}(1/2)^{\overline{D}} \right] + \overline{D} \left(\frac{1}{4}\right)^{\overline{D}} \\
&= 1 - \left(\frac{1}{2}\right)^{\overline{D}-1} - \frac{1}{3} + \frac{1}{3} \left(\frac{1}{4}\right)^{\overline{D}-1} - \frac{1}{3} \left[1 - (1/4)^{\overline{D}} - 3\overline{D}(1/4)^{\overline{D}} \right] \\
&\quad + \frac{2}{2\overline{D}} \left[1 - (1/2)^{\overline{D}} - \overline{D}(1/2)^{\overline{D}} \right] + \overline{D} \left(\frac{1}{4}\right)^{\overline{D}} \\
&= 1 - 2 \left(\frac{1}{2}\right)^{\overline{D}} - \frac{1}{3} + \frac{4}{3} \left(\frac{1}{4}\right)^{\overline{D}} - \frac{1}{3} + \frac{1}{3} \left(\frac{1}{4}\right)^{\overline{D}} + \overline{D} \left(\frac{1}{4}\right)^{\overline{D}} + 2 \left(\frac{1}{2}\right)^{\overline{D}} - 2 \left(\frac{1}{4}\right)^{\overline{D}} - 2\overline{D} \left(\frac{1}{4}\right)^{\overline{D}} + \overline{D} \left(\frac{1}{4}\right)^{\overline{D}} \\
&= \frac{1 - (1/4)^{\overline{D}}}{3}
\end{aligned}$$

A.2 Derivation of $P(n)$ and $P(n|\overline{H}, \overline{D})$

In the coinflip model Def. 15, every node of a tree can be thought of flipping a coin. Every tree can be presented as a binary string such that unity means “add child and ” and zero “go back”. If the tree contains N nodes, the corresponding string will consist of $2N$ bits: N 1’s and N 0’s. Conversely, any binary string of N 1’s and N 0’s is not a tree. There are $\binom{2N}{N} = (2N)!/(N!)^2$ such strings.

First we wish to derive the distribution of total node counts $P^*(n)$ assuming no constraints in child count D or height H . The distribution can be derived recursively as follows. Let us consider first all the nodes except for the root, returning to the root in (46). The probability of subtree sizes rooted to such a node is $P(n)$. Given an n , the node can be thought of “swallowing” one out of the $n-1$ and “delegating” the rest $n-1$ to its minor siblings. The precondition for such delegation is that such “delegates” exist:

at least one child exists with the probability of “heads”, one half. The same applies for siblings. The probability of “tails” is also half, but for readability of the following formulae let us denote $P(\text{“heads”}) = \frac{\overrightarrow{1}}{2}$ and $P(\text{“tails”}) = \frac{\overrightarrow{1}}{2}$

$$P(n) = \frac{\overrightarrow{1}}{2} P(n-1) \frac{\overrightarrow{1}}{2} + \sum_{i=1}^{n-2} \frac{\overrightarrow{1}}{2} P(n-1-i) \frac{\overrightarrow{1}}{2} P(i) + \frac{\overrightarrow{1}}{2} \frac{\overrightarrow{1}}{2} P(n-1), \quad (39)$$

where the first term delegates all the $n - 1$ nodes to the descendants, the middle term delegates $n - 2, n - 1, \dots, 1$ to descendants and $1, 2, \dots, n - 2$ to siblings, respectively, and the third term delegates all the $n - 1$ to the siblings. Defining $P(0) = 1$ for convenience allows writing

$$P(n) = \frac{1}{4} \sum_{i=0}^{n-1} P(n-1-i)P(i) \quad (40)$$

The root is the only exception by having no siblings, leaving the $n - 1$ nodes to the descendants. For the root and thus for the whole tree

$$P^*(n) = \frac{1}{2}P(n - 1). \quad (41)$$

Given a node with maximally $d-1$ *minor siblings* and maximal height h , the probability corresponding to (40) is now $P(n|d, h)$. While in the unconstrained coinflip trees every node except for the root has the same distribution of minor siblings and descendants, now the constraints imply several cases needing separate treatment. The cases as depicted in Fig. 25 with references to the following formulae.

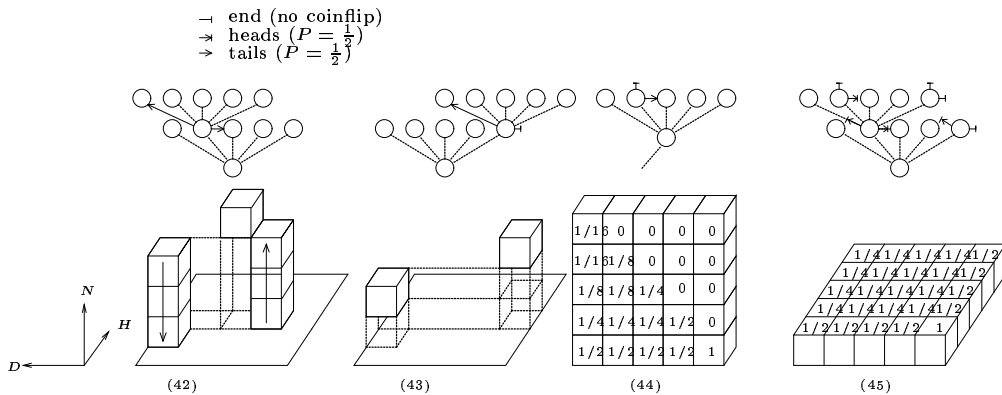


Figure 25: Derivation of $P(n|\overline{H}, \overline{D})$.

The “independent” case corresponds to (40): there exist at least one sibling and at least

one child for the focused node. The analogous counterpart for (40) is

$$P(n|d, h) = \frac{1}{4} \sum_{h=0}^{n-1} P(n-1-h|\overline{D}, h-1)P(h|d-1, h) \quad (42)$$

When the focused node has only children but no minor siblings (as it is the \overline{D} th child of its parent),

$$P(n|1, H) = \frac{1}{2}P(n-1|\overline{D}, h-1). \quad (43)$$

When the focused node has no children (as its depth is \overline{H}) but at least one minor sibling,

$$P(n|d, 0) = \begin{cases} \left(\frac{1}{2}\right)^n, & \text{if } n < d \\ \left(\frac{1}{2}\right)^{n-1}, & \text{if } n = d \\ 0 & \text{if } n > d \end{cases} \quad (44)$$

Also $P(1|d, h)$ is needed for initializing the calculations. As $n = 1$, the recursion should stop in both possible directions (siblings and children). There are four separate cases depending on whether coin flipping is needed or not in those directions.

$$P(1|d, h) = \begin{cases} 1, & \text{if } (d=1) \text{ and } (h=0), \\ \frac{1}{2}, & \text{if } (d=1) \text{ and } (h>0), \\ \frac{1}{2}, & \text{if } (d>1) \text{ and } (h=0), \\ \frac{1}{4}, & \text{if } (d>1) \text{ and } (h>0). \end{cases} \quad (45)$$

The analogous counterpart for (41) is

$$P^*(n|\overline{D}, \overline{H}) = \frac{1}{2}P(n-1|\overline{D}, \overline{H}-1). \quad (46)$$

The curves for $P^*(n)$ and $P^*(n|\overline{D} = 5, \overline{H} = 5)$ are plotted in Fig. 27, p. 80.

A.3 Sample data set

The data set used the experiments of Sec. 6.3 consists of 1000 coinflip trees with node size $N > 1$, maximum child count $\overline{D} = 5$, and maximum height $\overline{H} = 5$. The set is shown in Fig. 26.

The theoretical distribution of total vertex count, $P^*(n|\overline{D}, \overline{H})$ was derived in Sec. A.2 The distribution $\hat{P}^*(n|5, 5)$ of the 1000 sample trees is plotted together with the theoretical distribution $P^*(n|5, 5)$ in Fig. 27. The curve of the unconstrained distribution $P^*(n) = P^*(n|\infty, \infty)$ is included for comparison.

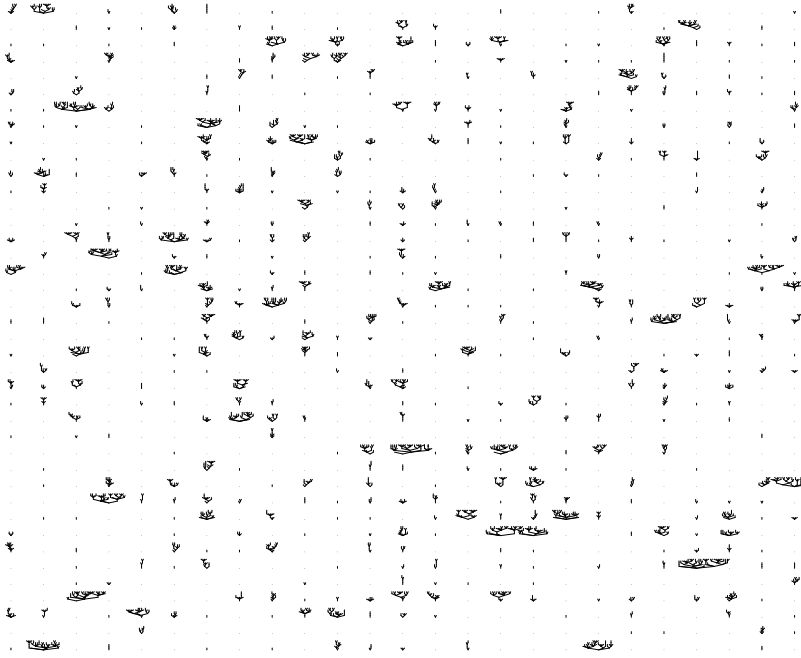


Figure 26: The set of 1000 coinflip trees used in the experiments.

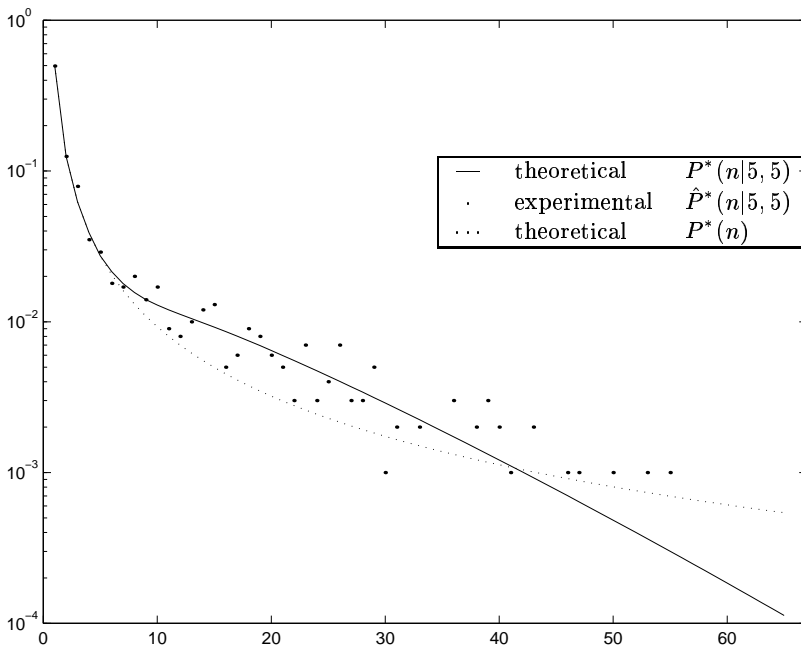


Figure 27: Theoretical and experimental distribution of $P^*(n|5,5)$.