

Matching and Learning in Trees

木の照合と学習

Tetsuji Kuboyama

久保山 哲二

Doctoral dissertation, 2007

Department of Advanced Interdisciplinary Studies
Graduate School of Engineering
The University of Tokyo



A Doctoral Dissertation

submitted to

Department of Advanced Interdisciplinary Studies,
Graduate School of Engineering,
The University of Tokyo,

in partial fulfillment of the requirements
for the degree for Doctor of Engineering

Typeset with p \LaTeX 2 ϵ , PSTricks (Ver. 1.10), and X γ -pic (Ver. 3.7)

Copyright © 2007 Tetsuji Kuboyama

Abstract

This thesis presents a unified understanding of edit-based approaches to approximate tree matching and introduces new facts on the subject. It also provides a broad view of kernel-based learning methods for trees, and proposes novel methods based on this view.

These contributions have a wide range of applications to pattern matching, computational biology, and many other areas of computer science. As an example, this thesis includes an application to computational biology to demonstrate the effectiveness of a novel learning method developed in this work.

This thesis is divided into two parts. The first part is devoted to a detailed analysis of the classes of edit-based approximate tree matching such as the tree edit problem and the alignment problem for trees. We focus on the notion of tree mapping in order to describe the semantics of approximate tree matching. We then establish an algebraic model of approximate tree matching. This algebraic model enables us to reveal unknown hidden relationships among a variety of edit-based approximate tree matching problems, which include the equivalence between two problems, the alignment problem and the less-constrained edit problem. Also, we give the tree mapping for the alignment of trees, which has been unknown for the past decade. In addition, we reveal a class hierarchy of edit-based approximate tree matching.

The second part focuses on tree kernels for kernel-based learning of trees. We first show that existing tree kernels are formulated by counting functions of tree mappings from the unified view based on the class hierarchy of approximate tree matching established in the first part. In addition, we develop two novel tree kernels more flexible than known tree kernels, and show that the counting function for the alignment of trees does not satisfy a requirement for a tree kernel. The last half of the second part addresses the development of a faster tree kernel without sacrificing its learning performance. We propose two simple tree kernels, a *spectrum tree kernel* and its variant, a *gram distribution kernel*. The effectiveness of these methods are demonstrated by applying them to a problem in computational biology.

Keywords: approximate pattern matching, tree edit distance, alignment of trees, machine learning, kernel methods, tree kernels, convolution kernel, glycans

Acknowledgments

I owe immeasurable debts of gratitude to so many people who have supported me along the way. I would especially like to thank my thesis Committee Chair, Hiroshi Yasuda. He has been a generous and superb mentor throughout this work, and gave me an opportunity to have many valuable experiences and to encounter with fabulous people throughout the Information Security Project led by him. I am very grateful for having an exceptional committee for my thesis and wish to thank the other committee members, Koichi Hori, Mina Akaishi, Tetsuo Shibuya, and Terumasa Aoki for sparing their precious time and offering valuable suggestions.

Special thanks to Kilho Shin, who could not have been more supportive of me throughout this entire work as a coworker and a reliable friend. His keen logicity and superb mathematical intuition often shed light on my conjectures for achieving tangible results. Thanks also to Koichi Hirata, who is a highly productive researcher with a strong commitment to excellence, for sharing his vast expertise in theoretical computer science, and for encouraging me constantly to write sound and persuasive papers; Hisashi Kashima and Kiyoko Flora Aoki-Kinoshita for their support in their areas of expertise, kernel-based learning and glycomics respectively. It has been a distinct pleasure to have had the opportunity to work with such leading researchers in their fields.

I would also like to express my gratitude to: Masaru Kitsuregawa for his patient support and encouragement, and for giving me the opportunity to continue to study at the Institute of Industrial Science (IIS) of the University of Tokyo; All the colleagues at the Computer Center in IIS, Hitomi Fukushima, Hiroshi Hayashi, Kiyomitsu Hirabaru, and Tsuneo Suzuki, who allowed me to concentrate on writing this thesis; Hidetoshi Yokoi, who has made efforts to provide me with such a pleasant working atmosphere at the Center for Collaborative Research of the University of Tokyo; Makoto Amamiya for his endurance while I had been drifting like a jellyfish aimlessly in the depthless abyss without realizing what an ample research environment had been provided by him; Akira Yasuhara for his generosity, a lot of helpful discussions about graph theory, dedicated support, and constant encouragement; Hiroki Arimura, who first appreciated my work in its immature stage, and has since then always been a catalyst for advancing this work; Tetsuhiro Miyahara, whose work on data mining of semistructured data motivated me to start on a part of this work when I almost drifted away from research activities while being swamped with network and database administration tasks.

I have had the tremendous fortune to encounter and get (re)acquainted with fabulous researchers active in the front lines in their fields through this work. I would like to return my grateful acknowledgment to Tatsuya Akutsu, Gabriel Valiete, Masafumi Yamashita, Jesper Jansson, Makoto Haraguchi, Sachio Hirokawa, Kazuki Joe, Shin-ichi Minato, Jingde Cheng, Haruo Yokota, Tamás Horváth, Takeaki Uno, Shin-ichi Nakano, and Alessandro Moschitti for helpful discussions, support, and encouragement.

I would also like to warmly thank all of the people who helped or encouraged me in various ways during this work: Kunihiko Mabuchi, Mari Yasuhara, Kyoko Shin, Juan Carlos Letelier, Tomoyuki Uchida, Takuya Kida, Rin-ichiro Taniguchi, Akira Yamamoto, Tsunenori Mine, Naoyuki Tsuruta, Ryuzo Hasegawa, Hiroshi Fujita, Miyuki Koshimura, Eisuke Ito, and Nobuhito Ohkura. Thanks to Bruce Furmedge and Christina Stephens, who read part of this thesis in draft form, for suggestions and corrections in English. Thanks also to Shelley Protte for her constant support in the English language and encouragement as a congenial friend.

Finally, I would like to thank my family: Chiaki and Shigeko for their sacrifice and constant encouragement; Keiichi for being a good confidant; Yuko for her dedicated and understanding support throughout this work; Tomoaki for his carefree smiles.

Contents

Abstract	i
Acknowledgments	iii
1 Introduction	1
1.1 Background and Objectives	1
1.1.1 Matching in Trees	1
1.1.2 Learning in Trees	2
1.2 Organization	3
1.3 Main Results	3
1.4 Conventions Used in This Thesis	4
I Matching in Trees	7
2 Approximate Tree Matching	9
2.1 Distance and Metric	9
2.2 Approximate String Matching	9
2.2.1 Strings	10
2.2.2 String Edit Distance	10
2.2.3 Approximate Common Subsequence Problem	12
2.2.4 Approximate Common Supersequence Problem — Alignment	13
2.2.5 Operational Definition	14
2.2.6 Declarative Definition	16
2.2.7 Approximation of String Edit Distance	17
2.3 Basic Notation for Trees	19
2.3.1 Rooted Trees	19
2.3.2 Syntactic Representation of Ordered Trees and Forests	22
2.3.3 Tree Traversals	23
2.4 Tree Edit Distance — Tai Distance	24
2.4.1 Edit Operations	24
2.4.2 Tree Edit Problem	26
2.4.3 Algorithms for Ordered Trees	26
2.4.4 Algorithms for Unordered Trees	31
2.5 Tree Mappings	32
2.5.1 Tree Mapping based Distance	32
2.5.2 Tai Mapping — Declarative Definition of Tai Distance	34
2.5.3 Approximate Common Subforest Problem	36
2.6 Variants of The Tree Edit Problem	37
2.6.1 Unit Costs	37
2.6.2 Largest Common Subforest Patterns	38
2.7 Alignment of Trees and Alignment Distance	38
2.7.1 Operational Definitions	38
2.7.2 Approximate Common Supertree Problem	40
2.7.3 Algorithms for Ordered Trees	41
2.7.4 Algorithms for Unordered Trees	45

2.7.5	Alignable Mappings	46
2.8	Structure Sensitive Distance	47
2.8.1	Structure-Preserving Distance	47
2.8.2	Strongly Structure-Preserving Distance	50
2.8.3	Lu Distance	50
2.8.4	Constrained Distance — Isolated-Subtree Distance	51
2.8.5	Structure-Respecting Distance	55
2.8.6	Less-Constrained Distance	55
2.9	Subtree Isomorphism based Distance	57
2.9.1	Top-Down Distance — LCST Problem	57
2.9.2	Bottom-Up Distance	61
2.10	Related Work	61
2.10.1	Hierarchical View of Tree Mappings	61
2.10.2	Tree Inclusion Problem	62
2.10.3	Additional Edit Operations	62
2.10.4	Gap Costs	63
2.10.5	Local Similarity between Trees	63
2.10.6	Approximation of Tree Edit Distance	63
2.10.7	Edit Distance between Graphs	64
2.11	Summary	64
3	Theoretical Foundation of Approximate Tree Matching	67
3.1	Preliminaries	67
3.2	Tree Homomorphism	68
3.3	Tree Embedding	70
3.4	Insertion	73
3.5	Tree Contraction	74
3.6	Deletion	78
3.7	Duality between Embedding and Contraction	79
3.8	Summary	80
4	Relationship Analysis among Tree Edit Distance Measures	81
4.1	Constrained and Structure-Respecting Mappings: $C_{ST} = SR$	81
4.2	Structure-Preserving and Constrained Mappings: $SP \supseteq C_{ST}$	82
4.3	Strongly Structure-Preserving and Constrained Mappings: $SP^b = C_{ST}$	83
4.4	Less-Constrained Mapping Revised	84
4.5	Constrained and Less-Constrained Mappings: $C_{ST}^\# \supseteq C_{ST}$	85
4.6	Alignable and Less-Constrained Mapping: $ALN = C_{ST}^\#$	85
4.6.1	Algebraic Formulation of Alignable Mappings	85
4.6.2	Equivalence between Alignable and Less-Constrained Mappings	88
4.6.3	Property of Alignable Mappings	91
4.7	Semi-Accordant Mappings: $Acc^\# = C_{ST} = SP^b = SR$	91
4.8	Accordant and Lu Mappings: $Acc \supseteq Acc^* = LU$	92
4.8.1	Closure of Tree Mappings	93
4.9	Summary	94
II	Learning in Trees	99
5	Kernel-based Learning for Trees	101
5.1	Support Vector Machines	101
5.2	Kernel Methods	102
5.3	Hausser's Convolution Kernels	102
5.3.1	Gap-Weighted String Kernel	104
5.3.2	Spectrum Kernel for Strings	105
5.4	Tree Kernels	105
5.4.1	Parse Tree Kernel	106
5.4.2	Labeled Tree Kernel	106

5.4.3	Labeled Tree Kernel with Elastic Structure Matching	107
5.4.4	String Kernel for Trees	109
5.5	Summary	111
6	Mapping Kernels for Trees	113
6.1	Recursive Expressions of Counting Functions	113
6.1.1	Mapping-based Similarity between Forests	113
6.1.2	Counting Function for Tai Mappings	114
6.1.3	Template of Counting Function for Subclasses of Tai Mappings	115
6.2	Positive Semidefiniteness of Counting Functions	119
6.3	Summary	123
7	Spectrum Kernels for Trees	125
7.1	Tree q -Grams	125
7.2	Spectrum Kernel for Trees	129
7.3	q -Gram Distance for Trees	131
7.4	Gram Distribution Kernel	132
7.5	Summary	134
8	Application to Glycan Classification	137
8.1	Glycan Data	137
8.2	Experimental Results	138
8.2.1	Computation Time	138
8.2.2	Glycan Data Classification by Spectrum Tree Kernel	138
8.2.3	Predictive Accuracy	139
8.2.4	Motif Extraction by Gram Distribution Kernel	140
8.2.5	Results and Discussion	140
8.3	Summary	142
9	Conclusion and Future Work	145
9.1	Conclusion	145
9.2	Future Work	146
	Bibliography	149
	Index	158

List of Figures

2.1	Dynamic programming algorithm for string edit distance	15
2.2	Example of a trace	16
2.3	A tree T and a forest of T induced by U	21
2.4	The composite of two forests F_1 and F_2	22
2.5	Tree $T = v(F)$	22
2.6	Left-to-Right Preorder and Postorder numberings of an ordered tree	24
2.7	Examples of the three elementary edit operations	25
2.8	Left-to-right postorder numbering of nodes	29
2.9	Two extreme examples in Zhang-Shasha's algorithm	29
2.10	Tai mapping	34
2.11	Non-Tai mapping	35
2.12	Approximate Common Subforest	37
2.13	Alignment of trees	40
2.14	Alignable mapping and its aligned tree	40
2.15	Another alignable mapping and its aligned tree	40
2.16	Non-alignable mapping	41
2.17	Alignment problem as an edit problem	41
2.18	Tai edit problem	41
2.19	Recurrences for computing alignment distance for ordered trees	42
2.20	$D_T(\emptyset, v_2(F_2)) + \min_{T \in F_2} \{D_T(v_1(F_1), T) - D_T(\emptyset, T)\}$ in Eq.(2.4a)	42
2.21	$D(T_1^1 \bullet \cdots \bullet T_1^{i-1}, F_2) + D(T_1^i \bullet \cdots \bullet T_1^m, F_2)$ in Eq.(2.4c)	42
2.22	Two optimal alignable mappings	45
2.23	Two aligned trees obtained by using the same alignment	45
2.24	Counterexample to transitivity in alignable mappings	46
2.25	Non-metricity of alignable distance	46
2.26	Structure-preserving mapping	48
2.27	Non-structure-preserving mapping	49
2.28	Asymmetry of structure-preserving mapping	49
2.29	Constrained mapping	52
2.30	Non-constrained and structure-preserving mapping	52
2.31	Recurrences for computing constrained distance for ordered trees	53
2.32	Feature of constrained and less-constrained mappings	56
2.33	Recurrences for computing less-constrained distance for ordered trees	57
2.34	Two input trees in computing top-down distance	59
2.35	Edit graph for two ordered trees S and T	60
2.36	Top-down mappings	60
2.37	Bottom-up mapping	62
3.1	An ordered tree homomorphism f from S to T	69
3.2	Tree isomorphisms for unordered trees.	70
3.3	An ordered embedding f from S to T	71
3.4	An example of the property in Proposition 3.19.	72
3.5	Corollary 3.20 cannot be extended to the 4-node case.	73
3.6	The definition of contraction	75
3.7	A contraction f from S to T	75
3.8	The duality between contraction f and embedding g	79

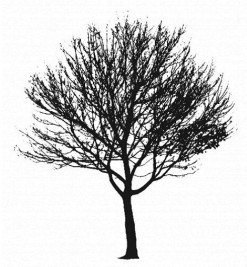
3.9	Algebraic formulation of approximate tree matching	80
4.1	Left-to-right preorder and sibling order	82
4.2	Folding two subtrees $T(v)$ and $T(w)$	86
4.3	Accordant mapping	92
4.4	Closure of a tree mapping	94
4.5	A Tai mapping with no closure	94
4.6	Class hierarchy of tree mappings	96
5.1	Two ordered trees for the labeled tree kernels	107
5.2	Two ordered trees for the elastic tree kernels	108
5.3	What is the feature space in the elastic tree kernel?	109
5.4	Two ordered trees for Vishwanathan-Smola's tree kernel	110
5.5	Expressive power of the string kernel for trees due to Vishwanathan and Smola	110
6.1	A tree mapping in \mathcal{M}_3 : $M = M[1, 2] \cup M[3, 3] \cup M[4, 5] \cup M[7, 6]$	116
6.2	A counterexample to positive semidefiniteness	120
7.1	An ordered tree with postorder numbering	126
7.2	4-grams (P_1, abab) , (P_2, baab) , and (P_3, abab)	127
7.3	The table <i>count</i> for $q = 4$ and $\max D = 3$	128
7.4	The table <i>shift</i> for $q = 4$ and $\max D = 3$	128
7.5	The trees T_1 and T_2 in Example 7.6	131
7.6	The relationship among d, j, k and $ UP_r(p) $ for a q -gram P_k^q	133
8.1	The running time for computing \mathbf{K}_t and \mathbf{K}_q	139
8.2	Area under the ROC curve	139
8.3	The performances of the SVM classifier for the gram distribution kernel and the layered trimer kernel	140
8.4	The distributions of feature scores	141

List of Tables

2.1	Three views of string edit distance	17
2.2	Computational complexity of Tai edit problem for ordered trees	27
2.3	Computational complexity of Tai edit problem for unordered trees	31
2.4	Algorithms for Tai distance for unordered trees	31
2.5	Some classes of approximate tree matching	37
2.6	Computational complexity of alignment problem	38
2.7	Computational complexity of structure sensitive distance problems	47
2.8	Computational complexity of tree inclusion problem	62
4.1	Properties of tree mapping classes	95
4.2	Characteristic of tree mapping classes	97
5.1	Feature vectors in the labeled tree kernel	107
5.2	Feature vectors in Vishwanathan-Smola’s tree kernel	110
6.1	Kernels by counting functions of tree mappings	123
7.1	The depth sequence, the label sequence, and the parent sequence of T	126
7.2	The transition of $freq[j][k]$ and $label[d]$ in T_1 and in T_2	130
7.3	The 4-gram profiles of T_1 (left) and T_2 (right)	131
7.4	The transition of the table $freq$ in T	135
8.1	The data labels, and the number of each data set in the experiments	138
8.2	Features extracted by our method	142

List of Algorithms

2.1	String edit distance and traceback	14
2.2	Tree traversals	23
2.3	Zhang-Shasha's algorithm for Tai distance	28
2.4	Alignment distance for ordered trees	43
2.5	Constrained distance for ordered trees	54
2.6	Selkow's algorithm for top-down distance	58
2.7	Chawathe's algorithm for top-down distance	60
5.1	Labeled tree kernels	108
7.1	PSEQ	126
7.2	LABELGRAM	127
7.3	LABELGRAMDIST	133



Chapter 1

Introduction

This thesis is a comprehensive study of approximate pattern matching and machine learning for trees. A tree is a mathematical abstraction representing a hierarchical structure of information, which allows us to effectively access and maintain data. The expressive power of trees is superior to strings, and inferior to general graph structures. A string is regarded as a simple form of tree, i.e. a node-labeled tree with only one branch.

We already have the fertile field of stringology, the study of strings, which enjoys a lot of elegant and pragmatic methods along with the history of computer science for string pattern matching, string indexing, and so forth. Over the fertile field of stringology, the study of trees has also been carried out for many years. Actually, in some ways, the study of trees can be viewed as just as highly-developed and mature a field as stringology. So what contribution to this field above and beyond the constellation of existing work on trees can we make? In what follows, we discuss the motivation behind our work.

1.1 Background and Objectives

When considering algorithms on trees, they may remind us of a great number of data structures and algorithms such as binary search trees, red-black trees, and B-trees, suffix trees, range trees, and k -d trees (cf. [MS05]). All of the tree structured data used in these methods are internal representations for efficient manipulation or succinct organization of entities in the real world.

For example, suffix trees are used for indexing strings, and k -d trees used for partitioning d -dimensional space. These trees are obtained by reconstructing the structures of entities in the real world such as strings of DNA sequences or two-dimensional matrices of pixel images rather than directly reflecting the surface tree structures in the entities.

The main concern with these algorithms lies in how to efficiently construct and traverse such an internal tree structure, whereas a tree-to-tree comparison algorithm is required for dealing with the tree structured entities.

After all, there have not been urgent requirements for comparing tree structured entities until recently with a few exceptions such as phylogenetic trees. The amount of tree structured data, however, not derived from internal representations has dramatically increased in the past decade with the rapid growth of the Internet, and some research fields such as bioinformatics. For example, tree structured data such as HTML and XML are widely found in Web-based systems. Also, in the field of bioinformatics, we can see tree structured data such as RNA secondary structures and glycan structures. On top of that, due to some recent developments in graph theory, the tree decomposition algorithm enables us to view a complex graph structure as a tree-like structure.

1.1.1 Matching in Trees

Recently the problem of measuring the similarity of two trees has been a focus of researchers in various scientific fields such as computational biology [Aku00, Sak03, HTGK03, AS04], image analysis [TH03d,

TH02, TH03c, TH03e, TH03b, Tor04, BBP04, Ols05], pattern recognition [FG00, GB02], natural language processing [FRV04] and information extraction from Web pages [CD01, HK05, ZL05]. For example, the secondary structure of an RNA chain is an important factor for determining its functions. Since RNA secondary structures are often represented trees, measuring the similarity of the trees that represent the secondary structures of an unknown RNA chain and a known RNA chain could provide plenty of clues for us to guess the functions of the unknown RNA from knowledge of the known RNA.

Tree edit distance is the most widely accepted metric for measuring the difference or dissimilarity of trees. The tree edit distance between two trees is defined as the minimum cost of a series of elementary edit operations needed to transform the first tree into the second.

Tai presented an important correspondence between tree edit distance and *tree mapping* [Tai79]. Since then, tree mapping has been attracting the interest of researchers. The tree mapping between two trees is a set-theoretic description of the transformation from one tree to the other. Intuitively, a tree mapping describes *what* the transformation between two trees is by showing a set of node pairs, whereas a tree edit algorithm describes *how* to transform one tree into the other. Tree mapping allows us to understand and investigate tree edit distance in a qualitative and abstract way.

Other than the tree edit distance proposed by Tai (*Tai distance*), a variety of tree edit distance measures have been proposed in the past three decades. Following in the wake of Tai's result, most of these measures have been defined by using the notion of tree mapping in conjunction with the algorithms. For example, the algorithms for computing the *structure-preserving* [TT82, Tan93, TT88, Tan95], *constrained* [Zha95, Zha96], *structure-respecting* [Ric97], *less-constrained* [LST01], and *bottom-up* [Val01] distance measures were proposed according to the definitions using the notion of tree mapping, i.e. these measures have the corresponding tree mapping definitions. We refer to the tree mapping defining Tai distance as the Tai mapping, and we also refer to the other classes of tree mappings in the same way. In contrast, the tree mapping defining *alignment distance* remains unknown in spite of its significance since this distance measure [JWZ95] was proposed in 1995.

In response to these results, Wang and Zhang revealed a hierarchy of some of tree edit distance measures based on the analysis of tree mapping [WZ01]. Nevertheless, these measures are not exactly based on a common mathematical formalization. Consequently, less or more, they tend to be subject to ambiguity, redundancy and sometimes inaccuracy in their mathematical discussion. For example, the original definition of *less-constrained* mapping [LST01], which had been prevailing (e.g. presented in a widely cited survey on tree edit distance [Bil05], and some other publications such as [Höc05]), is incorrect. Also, the original definition of bottom-up mapping [Val01] is not consistent with the proposed algorithm. For another example, the equivalence of notions between the structure-preserving and the constrained mappings were mentioned in [Zha96] although both notions are not exactly equivalent. Also, the equivalence between the constrained and structure-respecting mappings were mentioned in [LST01] without any mathematical proofs.

Taking this observation into account, we aim to establish a theoretical foundation that could form a common basis for the study of approximate tree matching, and introduce new facts including relationships among a variety of tree edit distance measures.

1.1.2 Learning in Trees

The kernel method, a method of machine learning, provides a generic framework to address a variety of applications, and is being extensively studied [STC04]. The problems to which this method can be applied include the *classification problem*, i.e. the problem of determining the class to which a given instance belongs. In kernel methods, in order to design a classifier for trees is necessary to design a similarity measure (i.e. tree kernel) between two trees. This task is very similar to designing an algorithm for tree edit distance. The problem of computing tree edit distance is regarded as a combinatorial optimization problem of tree patterns while the problem of computing a tree kernel is regarded as a counting problem of tree patterns.

Since Collins and Duffy first proposed a tree kernel for parse trees [CD01], a variety of tree kernels have been proposed such as kernels for parse trees in natural language processing [ZAR03, CS04, SI05, Mos06], phylogenetic trees [Ver02], Prolog proof trees [PFR06], and glycans [HYHK04, HYN⁺05]. In this work, we focus on general-purpose kernels for labeled trees [VS02, KK02, KSK06a].

By theoretical analysis of existing tree kernels, a significant affinity between approximate tree matching and kernel-based learning for trees has emerged. This thesis reveals a hidden and important relationship between the two fields. Based on the relationship, we develop novel tree kernels more flexible than known

tree kernels.

From the practical point of view, we address the development of a faster tree kernel without sacrificing its learning performance. We propose two simple tree kernels, a spectrum tree kernel and its variant, a gram distribution kernel. The effectiveness of these methods are demonstrated by applying them to a problem in computational biology.

1.2 Organization

This thesis is divided into two parts. Part I is devoted to approximate tree matching, and divided into three chapters. Chapter 2 surveys a variety of conventional edit-based approaches to approximate tree matching with unifying mathematical formulation. Chapter 3 establishes a theoretical foundation for approximate tree matching in an algebraic way. In our formalization, we first introduce a very general mapping between trees, and call it a tree homomorphism. Starting with the notion of tree homomorphism, we tighten the tree mapping gradually to adjust it to existing edit operations. Chapter 4 reveals the relationship among existing tree edit distance measures by means of the formulation presented in the previous chapter.

Part II focuses on kernel-based machine learning for trees, and divided into four chapters. Chapter 5 surveys conventional kernel-based learning methods for trees. Chapter 6 proposes a novel tree kernel based on counting the number of tree mappings. Chapter 7 presents a simple and efficient tree kernel based on the notion of tree q -gram, and presents its variant. Chapter 8 shows the application of proposed tree kernels to the glycan classification problem. Finally, in Chapter 9, we conclude this thesis by discussing further issues to be addressed after the entire summary.

1.3 Main Results

Here, we list the main results presented in this thesis along with references to the relevant publications.

Part I. Matching in Trees

All the results presented in this part have been published in [KSM05, KSMY05].

Chapter 3. Theoretical Foundation of Approximate Tree Matching

- We have established a theoretical foundation of edit-based approaches to approximate tree matching, which bridges the gap between operational semantics and declarative semantics of tree edit distance.

Chapter 4. Relationship Analysis among Tree Edit Distance Measures

- We have proved that the alignment problem [JWZ95] is essentially equivalent to a variant of the tree edit problem called the less-constrained edit problem [LST01].
- We have identified the *tree mapping* condition for the alignment of trees, which had previously been unknown. Tree mapping provides definitions of various tree edit distance measures in a declarative way, in contrast to the operational way of conventional definitions. To the best of our knowledge, the mapping condition for the alignment of trees has been unknown, and the alignment of trees was defined only in the operational way in [JWZ95].
- We have shown that the condition of the less constrained mapping given by Lu *et al.* [LST01] does not relax the condition of the constrained mapping due to Zhang [Zha96]. In fact, we show that the condition due to Lu *et al.* [LST01] is identical to that of the constrained mapping. We revise it and give an originally intended condition of less-constrained mapping.
- We have proved that the constrained distance [Zha96] and the structure-respecting distance [Ric97] are equivalent, but structure-preserving distance [TT88] is a superclass of these two distance measures.

Part II. Learning in Trees

Chapter 6. Mapping Kernels for Trees

The results in this chapter have been published in [KSK06b].

- We have presented the counting functions for the four mapping classes, i.e. Tai, alignable, semi-accordant, and accordant mappings, in recursive form. These forms enable us efficient evaluation by dynamic programming.
- We have proved that the counting functions for the accordant, semi-accordant, and Tai mappings are positive semidefinite. In contrast, for the alignable mapping, a counterexample to positive semidefiniteness has been given. Hence, the counting functions for the Tai, semi-accordant, and accordant mappings are kernel functions whereas that for the alignable mapping is not. The accordant case proves that the elastic tree kernel [KK02] is a kernel function.

Chapter 7. Spectrum Kernels for Trees

The results in this chapter have been published in [OHKH05, KHOH06, KHAK⁺06, KHK⁺06, KHK⁺07].

- We have proposed two new kernel functions (similarity measures) for trees called *spectrum tree kernel* and *gram distribution kernel*. The spectrum tree kernel can be regarded as a natural extension of the spectrum string kernel. This string kernel counts the number of shared substrings of a fixed length q between two strings, while our kernels count the number of shared line-shaped connected graphs with a fixed number q of nodes occurring in two trees without sacrificing its efficiency as compared with the string case.

Chapter 8. Application to Glycan Classification

The results in this chapter have been published in [KHAK⁺06, KHK⁺06, KHK⁺07].

- The spectrum tree kernel and the gram distribution kernel have been applied to glycan structure analysis. Our results have shown that our kernel outperforms the layered trimer kernel of Hizukuri *et al.* [HYN⁺05] which is well tailored to glycan data while we do not adjust our kernel to glycan-specific properties.
- In addition, we have extracted specific features from various types of glycan data using our trained SVM. The results show that our kernel is more flexible and capable of finding a wider variety of substructures from glycan data.

1.4 Conventions Used in This Thesis

The following logical symbols are used for succinct descriptions.

- $\neg P$ not P .
- $P \wedge Q$ P and Q .
- $P \vee Q$ P or Q .
- $P \Rightarrow Q$ if P , then Q .
- $P \Leftrightarrow Q$ P if and only if Q .
- $\exists x$ there exists x .
- $\exists! x$ there exists exactly one x .
- $\forall x$ for all x .

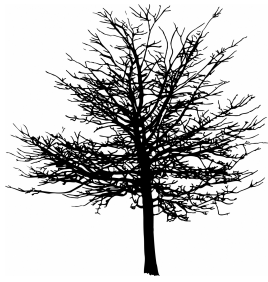
We use the following notational conventions throughout this thesis.

- $\mathbb{N} = \{1, 2, 3, \dots\}$ is the set of natural numbers.
- \mathbb{R} is the set of real numbers.
- \mathbb{R}_0^+ is the set of nonnegative real numbers.
- $\|\mathbf{x}\|_1$ is the ℓ_1 -norm on a vector $\mathbf{x} = (x_1, x_2, \dots, x_n)$ of real numbers defined as $\sum_{i=1}^n |x_i|$.

Also, we use the standard asymptotic *big O notation* such as $O(\cdot)$, $\Omega(\cdot)$, $\Theta(\cdot)$, $o(\cdot)$, $\omega(\cdot)$ (cf. [CLRS01, Chapter 3]).



The shaded paragraph beginning with this leaf icon designates an open problem or an incorrect assertion in prior work.



Part I

Matching in Trees

“For trees, you see, rather conceal themselves in daylight.
They reveal themselves fully only after sunset. I never *know* a tree,”

— Algernon Blackwood, *The Man Whom the Trees Loved*



Chapter 2

Approximate Tree Matching

Approximate pattern matching methods for trees have directly adopted many of the fundamental notions cultivated for strings. Hence we start with a cursory review on some of the basic notions used in approximate string matching. In particular, we concentrate on an edit-based approach, string edit distance, which is a prevailing approach as a common framework for measuring the difference or distance between two strings.

Subsequently, we introduce the basic notation for trees, and give a comprehensive survey on a variety of distance measures for trees presented in prior work. Toward a unified view of approximate tree matching, we point out confusion and inconsistency inherent in prior work.

2.1 Distance and Metric

Following the convention in the approximate tree matching field, we abuse the term *distance*, i.e. by distance we vaguely mean degree of dissimilarity, and then it does not necessarily mean *metric* in the mathematical sense.

Definition 2.1 (Metric) A *metric* on a set X is a mapping $d : X \times X \rightarrow \mathbb{R}$ that satisfies the following conditions:

1. $\forall x, y \in X \quad d(x, y) \geq 0$ (non-negativity),
2. $\forall x, y \in X \quad d(x, y) = 0 \iff x = y$ (equality),
3. $\forall x, y \in X \quad d(x, y) = d(y, x)$ (symmetry),
4. $\forall x, y, z \in X \quad d(x, z) \leq d(x, y) + d(y, z)$ (subadditivity, or triangle inequality).

The pair (X, d) is referred to as a *metric space*.

Note that the first condition, non-negativity, is derived from the other three conditions. If the second condition (equality) is replaced with the condition

$$2' \quad \forall x \in X \quad d(x, x) = 0,$$

then the mapping is called *pseudometric*.

2.2 Approximate String Matching

First, we introduce the conventional definition of string edit distance (cf. [Gus97, Chapter 11], [NR02, Chapter 6], [CR02, Chapter 12], [Rah07]). The definition is given in an operational way, i.e. the string edit distance is defined by describing *how* to compute it by applying edit operations. We refer to this way of definition as the *operational definition*. Next, we show two viewpoints of string edit distance:

- approximate common subsequence problem, and
- approximate common supersequence problem (alignment problem).

Although these two viewpoints are a mathematically equivalent problem for strings, they lead to two mathematically distinct problems for trees in contrast to strings.

In addition, we give an alternative definition by describing *what* the string edit distance is without using edit operations, which we refer to as the *declarative definition*. This type of definition plays an important role in theoretical analysis of tree edit distance.

We also review some approximation algorithms for string edit distance.

2.2.1 Strings

A finite nonempty set of symbols is called an *alphabet*, denoted by Σ . A *string* over Σ is a finite series of elements of Σ . We represent a string x consisting of a series of symbols a_1, a_2, \dots, a_n as

$$x = a_1 a_2 \cdots a_n \quad (a_i \in \Sigma \text{ for } i \in \{1, \dots, n\}).$$

The length of string x , denoted by $|x|$, is the number of symbols in x . For two strings $x = a_1 \cdots a_m$ ($a_i \in \Sigma$ for $i \in \{1, \dots, m\}$) and $y = b_1 \cdots b_n$ ($b_j \in \Sigma$ for $j \in \{1, \dots, n\}$), we define a *concatenation* of x and y as

$$x \cdot y = a_1 \cdots a_m b_1 \cdots b_n.$$

The *null string*, denoted by ε , is a string with length 0. In particular, for any string x ,

$$\varepsilon \cdot x = x \cdot \varepsilon = x.$$

We define Σ^n as follows

$$\Sigma^n = \begin{cases} \{\varepsilon\} & \text{if } n = 0, \\ \{x \cdot a \mid x \in \Sigma^{n-1} \wedge a \in \Sigma\} & \text{if } n \geq 1. \end{cases}$$

By Σ^* we denote the set of all strings over Σ , i.e.

$$\Sigma^* = \bigcup_{n \geq 0} \Sigma^n.$$

The i -th symbol in a string x is denoted by $x[i]$. For a string $x = a_1 \cdots a_n$, a *substring* of x is a string $a_i \cdots a_j$ such that $1 \leq i \leq j \leq n$, denoted by $x[i..j]$. Let us define $x[i..j] = \varepsilon$ for $i > j$. A *subsequence* of x is a string $a_{i_1} a_{i_2} \cdots a_{i_k}$ such that $1 \leq i_1 < i_2 < \cdots < i_k \leq n$.

Note that for a string x , all the symbols in a substring of x need to be consecutive in x while the symbols in a subsequence is not necessarily. For example, “test” is a subsequence of “tapestry” but not a substring, while “tape” is a subsequence and a substring.

2.2.2 String Edit Distance

The *string edit distance* probably originated from *Levenshtein distance* [Lev66]. The Levenshtein distance between two strings is the number of deletions, insertions, or replacements of symbols required to transform the first string into the second. Later, the string edit distance was generalized to consider the cost of edit operations according to the relevant symbols to be edited [WF74].

Edit Operations

The string edit distance between two strings is defined as the minimum cost of elementary edit operations required to transform one string into the other.

The set of elementary *edit operations* on a string consists of the following three operations.

Replacement of a symbol in a string by a new symbol in Σ .

Deletion of a symbol from a string.

Insertion of a symbol in Σ into a string.

In order to estimate the cost of edit operations, we consider the cost factor of each edit operation, which we call an *edit signature* denoted as follows:

- “ $a \mapsto b$ ” for the replacement of a symbol a in the first string by a symbol b in the second,
- “ $- \mapsto b$ ” for the insertion of a symbol b in the second string into the first,
- “ $a \mapsto -$ ” for the deletion of a symbol a from the first string,

where the symbol “ $-$ ” is not in Σ , and called the *gap symbol*. Let us abuse notation by referring to each edit operation as its edit signature although the edit signatures do not have enough information for actual edit operations if there is no confusion. In fact, an edit signature does not tell us about the position to be edited. This implies that we focus only on the transition of symbols for estimating the cost for edit operations applied to a string, and do not consider the positions of symbols in the string. We write $x \xrightarrow{e} y$ if we obtain a string y from a string x by applying an edit operation e .

Let d be a cost function of edit signatures, and we equate the two notations $d(a \mapsto b)$ and $d(a, b)$ as follows:

$$d(a \mapsto b) = d(a, b) \quad \text{for all } a, b \in \Sigma \cup \{-\},$$

where we assume that $d : \Sigma \cup \{-\} \times \Sigma \cup \{-\} \rightarrow \mathbb{R}$ is a metric.

Example 2.2 Consider the string “string” with the alphabet $\Sigma = \{\mathbf{a}, \mathbf{b}, \dots, \mathbf{z}\}$.

- string $\xrightarrow{i \mapsto o}$ strong (replacement of “i” by “o”)
- string $\xrightarrow{r \mapsto -}$ sting (deletion of “r”)
- string $\xrightarrow{- \mapsto a}$ staring (insertion of “a”)

String Edit Problem

We here forbid edit operations with the edit signatures $a \mapsto b$ such that $a = b$. If a series of edit operations $E = \langle e_1, \dots, e_n \rangle$ ($n \geq 1$) transforms a string x into a string y , there exists a series of strings $\langle x_0, \dots, x_n \rangle$ such that $x_0 = x$, $x_n = y$, and the i -th edit operation $e_i = (a_i \mapsto b_i)$ transforms x_{i-1} into x_i for $i \in \{1, \dots, n\}$, i.e.

$$x = x_0 \xrightarrow{e_1} x_1 \xrightarrow{e_2} \dots \xrightarrow{e_n} x_n = y.$$

If $x = y$, then we define $n = 0$. The cost function $\text{cost}(\cdot)$ for a series of edit operations $E = \langle e_1, \dots, e_n \rangle$ is derived from the total cost for the elementary edit operations as follows:

$$\text{cost}(E) = \sum_{i=1}^n d(e_i).$$

If $n = 0$, then we define $\text{cost}(E) = 0$. We refer to the series of edit operations E as the *edit script*. Now let us denote the set of all possible edit scripts to transform x into y by $\mathcal{E}(x, y)$. Then the formal definition of string edit distance is given as follows.

Definition 2.3 (String Edit Distance) The edit distance between two strings x and y is defined as follows:

$$\mathbf{D}^{\text{EDIT}}(x, y) = \min_{E \in \mathcal{E}(x, y)} \text{cost}(E).$$

We refer to an edit script with the minimum cost as an *optimal edit script*. Note that the optimal edit script is not necessarily determined uniquely. The problem of computing the edit distance between two strings along with an optimal edit script is called the *string edit problem*. It is known that for any two strings x and y , the edit distance $\mathbf{D}^{\text{EDIT}}(x, y)$ is a metric if $d(a, b)$ is a metric for any $a, b \in \Sigma \cup \{-\}$.

Recall that the Levenshtein distance between two strings is the number of deletions, insertions, or replacements of symbols required to transform the first string into the second. We refer to Levenshtein distance as string edit distance with *unit costs* or simply *unit-cost edit distance*. More precisely, in this distance measure, the cost function $d : (\Sigma \cup \{-\}) \times (\Sigma \cup \{-\}) \rightarrow \mathbb{R}$ is defined as

$$d(a, b) = \begin{cases} 0 & \text{if } a = b, \\ 1 & \text{if } a \neq b. \end{cases}$$

We denote by $\mathbf{D}_1^{\text{EDIT}}(x, y)$ the unit-cost edit distance between two strings x and y .

Example 2.4 Let $\Sigma = \{A, C, G, T\}$. Now considering the unit-cost edit distance between two strings “ACTC” and “ACGT,” we can transform the first string into the second with minimum cost 2 as follows.

- $\text{ACTC} \xrightarrow{T \mapsto G} \text{ACGC} \xrightarrow{C \mapsto T} \text{ACGT}$, or
- $\text{ACTC} \xrightarrow{- \mapsto G} \text{ACGTC} \xrightarrow{C \mapsto -} \text{ACGT}$.

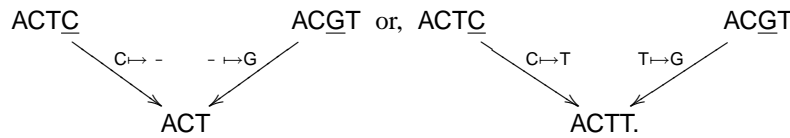
Hence the string edit distance between “ACTC” and “ACGT” is 2. ■

2.2.3 Approximate Common Subsequence Problem

There is an alternative view in defining string edit distance. In Definition 2.3, we apply all edit operations to the first string to obtain the second. In the alternative view, we use only *replacements* and *deletions* as the elementary edit operations, and define the string edit distance between two strings as the minimum cost of edit operations to transform two strings into a common third string. In other words, this problem is to find an approximate common subsequence shared by two strings with the minimum cost of edit operations without *insertions*. In this case, we permit the edit operation with edit signature $- \mapsto -$. This operation changes nothing and is called an *identity edit operation*.

The problem of computing the edit distance between two strings along with an approximate common subsequence of the minimum cost of edit operations is called the *approximate common subsequence problem*. It is easy to show that the approximate common subsequence problem is equivalent to the string edit problem in the computation of string edit distance since any deletion of a symbol a from the second string has its complementary operation, the insertion of a into the first string with the edit signature $- \mapsto a$. Obviously, the replacement of a symbol a in the first string by a symbol b in the second is equivalent to the replacement of b in the second by a in the first.

Example 2.5 We consider the same problem as in Example 2.4. Then, for example, the following two transformations give the minimum cost 2.



Hence the string edit distance between “ACTC” and “ACGT” is 2. ■

Longest Common Subsequence Problem

When we use only *deletions* as the elementary edit operations, and define the string edit distance as the minimum number of edit operations to transform given two strings into a common third string, the edit problem is said to be the *longest common subsequence problem* (LCS problem). In this problem, the cost function $d : \Sigma \cup \{-\} \times \Sigma \cup \{-\} \rightarrow \mathbb{R}$ is given as follows:

$$d(a, b) = \begin{cases} 0 & \text{if } a = b, \\ 1 & \text{if } a \neq b \text{ and } (a = "-" \text{ or } b = "-"), \\ \infty & \text{otherwise (replacement).} \end{cases}$$

Note that this cost function d is not a metric. Since each replacement can be replaceable by one deletion and one insertion, any cost of replacement greater than or equal to 2 leads to the same distance (if the cost of replacement is 2, d is a metric). We denote by $\mathbf{D}_{\text{LCS}}^{\text{EDIT}}(x, y)$ the edit distance with this cost function d between two strings x and y . The size of the longest common subsequence between two strings x and y is obtained by

$$\text{LCS}(x, y) = \frac{|x| + |y| - \mathbf{D}_{\text{LCS}}^{\text{EDIT}}(x, y)}{2}.$$

Note that, in general, the LCS problem is formalized by maximizing an edit *score* so that the length of a longest common subsequence is equal to the score. Many of elegant algorithms have been proposed [Gus97, Section 12.5], [BHR00].

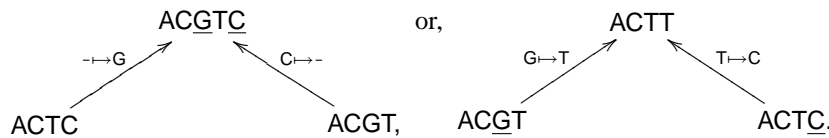
2.2.4 Approximate Common Supersequence Problem — Alignment

There is yet another alternative view in defining the string edit distance. First, let us define the notion of supersequence. For a string x , a *supersequence* of x is a string obtained by inserting arbitrary number of symbols into x . For example, “tapestry” is a supersequence of “test.”

In the alternative view, we use only *replacements* and *insertions* as the elementary edit operations, and define the string edit distance as the minimum cost of edit operations to transform two strings into a common third string. In other words, this problem is to find an approximate common supersequence shared by two strings with the minimum cost of edit operations without *deletions*. We permit the identity edit operation as in the approximate common subsequence problem.

The problem of computing the edit distance between two strings along with an approximate common supersequence of the minimum cost of edit operations is called the *approximate common supersequence problem*. As in the case of the approximate common subsequence problem, it is easy to show that the approximate common supersequence problem is equivalent to the string edit problem in the computation of the string edit distance since any insertion of a symbol a into the second string has its complementary operation, the deletion of a from the first string with the edit signature $a \mapsto -$.

Example 2.6 We consider the same problem as in Example 2.4. Then, for example, the following two transformations give the minimum cost 2.



Hence the string edit distance between “ACTC” and “ACGT” is 2. ■

Alignment Problem

Once we obtain an approximate common supersequence, we can align two strings according to the supersequence. For example, consider the two strings “ACTC” and “ACGT,” and an (approximate) common supersequence “ACGTC.” We have the following alignment.

First string	A	C		T	C
Approximate common supersequence	A	C	G	T	C
Second string	A	C	G	T	

In general, the *alignment* of two strings is depicted, without showing an approximate common supersequence, by padding out each string with the *gap symbol* “-” not in Σ to align each symbol at the same column.

First string	A	C	-	T	C
Second string	A	C	G	T	-

We refer to an alignment corresponding to the edit distance as an *optimal alignment*. Note that an optimal alignment is not necessarily unique. More formally, we can state the definition of the alignment of strings as follows.

Definition 2.7 (Alignment of Strings) An *alignment* of two strings x and y is obtained by the following two steps:

1. Insert gap symbols “-” into x and y so that the following two conditions are satisfied:
 - (i) two resulting strings x' and y' have the same length of n , i.e. $|x'| = |y'| = n$.
 - (ii) $x'[i] = y'[i] = \text{“-”}$ does not hold for any $i \in \{1, \dots, n\}$.
2. Collect the pairs of symbols at the same columns in order, i.e.

$$A = \langle (x'[1], y'[1]), (x'[2], y'[2]), \dots, (x'[n], y'[n]) \rangle.$$

The cost of an alignment A is defined as the sum of the costs of all pairs of aligned symbols :

$$\text{cost}(A) = \sum_{i=1}^n d(A[i]) = \sum_{i=1}^n d(x'[i], y'[i]),$$

Algorithm 2.1 String edit distance and traceback**procedure** EDITDISTANCE(x, y)Input: $x = a_1 \cdots a_m$ $y = b_1 \cdots b_n$ $D[0, 0] \leftarrow 0$ **for** $i \leftarrow 1$ **to** m **do** $D[i, 0] \leftarrow D[i - 1, 0] + d(a_i, -)$ **for** $j \leftarrow 1$ **to** n **do** $D[0, j] \leftarrow D[0, j - 1] + d(-, b_j)$ **for** $i \leftarrow 1$ **to** m **do****for** $j \leftarrow 1$ **to** n **do** $D[i, j] \leftarrow$

$$\min \begin{cases} D[i - 1, j - 1] + d(a_i, b_j) \\ D[i - 1, j] + d(a_i, -) \\ D[i, j - 1] + d(-, b_j) \end{cases}$$

return $D[m, n]$ **end****procedure** TRACEBACK($D[0..m, 0..n]$)Input: resulting array $D[0..m, 0..n]$ ofEDITDISTANCE(x, y) $i \leftarrow m; j \leftarrow n$ $x' \leftarrow \varepsilon; y' \leftarrow \varepsilon$ **until** $i = 0$ **and** $j = 0$ **do****if** $D[i, j] = D[i - 1, j - 1] + d(a_i, b_j)$ **then** $x' \leftarrow a_i \cdot x'; y' \leftarrow b_j \cdot y'$ $i \leftarrow i - 1; j \leftarrow j - 1$ **else if** $D[i, j] = D[i - 1, j] + d(a_i, -)$ **then** $x' \leftarrow a_i \cdot x'; y' \leftarrow "-" \cdot y'$ $i \leftarrow i - 1$ **else if** $D[i, j] = D[i, j - 1] + d(-, b_j)$ **then** $x' \leftarrow "-" \cdot x'; y' \leftarrow b_j \cdot y'$ $j \leftarrow j - 1$ **end until****return** (x', y') /* aligned strings */**end**

where $d : (\Sigma \cup \{-\} \times \Sigma \cup \{-\}) \setminus \{(-, -)\} \rightarrow \mathbb{R}$, and by abuse of notation we set $d((a, b)) = d(a, b)$. An *optimal alignment* is an alignment that minimizes the cost over all possible alignments. An *alignment distance* is the cost of an optimal alignment. We denote the set of all possible alignments between x and y by $\mathcal{A}(x, y)$. Then, the *alignment distance* between two strings x and y is given as follows.

$$D^{\text{ALN}}(x, y) = \min_{A \in \mathcal{A}(x, y)} \text{cost}(A).$$

Since each column in an alignment corresponds to a unique edit signature, edit distance is equivalent to alignment distance for strings.

2.2.5 Operational Definition

All these views so far lead to the following recurrences for computing string edit distance.

$$\begin{aligned} D(\varepsilon, \varepsilon) &= 0, \\ D(a \cdot x, \varepsilon) &= D(x, \varepsilon) + d(a, -), \\ D(\varepsilon, b \cdot y) &= D(\varepsilon, y) + d(-, b), \\ D(a \cdot, b \cdot y) &= \min \begin{cases} D(x, y) + d(a, b) \\ D(a \cdot x, y) + d(-, b) \\ D(x, b \cdot y) + d(a, -) \end{cases} \end{aligned}$$

where a and b are symbols from Σ , and $D(x, y)$ denotes the edit distance between two string x and y . We refer to the definition or algorithm describing *how* to compute string edit distance as the *operational definition* of it.

Needleman and Wunsch proposed an algorithm [NW70] (known as Needleman-Wunsch algorithm) for computing an optimal alignment by *dynamic programming* in the field of computational biology, while Wagner and Fischer introduced an algorithm for computing string edit distance [WF74]. Both algorithms have basically the same structure although Needleman-Wunsch algorithm computes an optimal alignment of two strings with maximum similarity score, i.e. the score is not necessarily a metric. Based on these two algorithms, in **Algorithm 2.1**, we show two algorithms for computing string edit distance (EDITDISTANCE) and an optimal alignment corresponding to the distance (TRACEBACK). The algorithm EDITDISTANCE computes the edit distance between two strings $x = a_1 a_2 \cdots a_m$ and $y = b_1 b_2 \cdots b_n$, and TRACEBACK computes two aligned strings according to the result of EDITDISTANCE. This procedure is called *traceback*. EDITDISTANCE runs in $\Theta(mn)$ time, and TRACEBACK runs in $\Theta(m + n)$ time.

In this algorithm, the edit distance problem for two strings is reduced to the shortest path problem in a kind of lattice graph called an *edit graph*. The edit graph is constructed according to the following definition.

Definition 2.8 (Edit Graph for Two Strings) Let $x = a_1a_2 \cdots a_m$ and $y = b_1b_2 \cdots b_n$ be strings. The *edit graph* of x and y is an edge weighted graph $G(x, y) = (V, E)$ such that

- the set of vertices V is $\{v_{(i,j)} \mid (i, j) \in \{0, \dots, m\} \times \{0, \dots, n\}\}$,
- the set of edges E consists of the following edges:
 - $(v_{(i-1,j-1)}, v_{(i,j)}) \in E$ with the weight $d(a_i, b_j)$ for $(i, j) \in \{1, \dots, m\} \times \{1, \dots, n\}$,
 - $(v_{(i-1,j)}, v_{(i,j)}) \in E$ with the weight $d(a_i, -)$ for $(i, j) \in \{1, \dots, m\} \times \{0, \dots, n\}$,
 - $(v_{(i,j-1)}, v_{(i,j)}) \in E$ with the weight $d(-, b_j)$ for $(i, j) \in \{0, \dots, m\} \times \{1, \dots, n\}$.

Given two strings x and y , each node of edit graph for x and y is represented as $D[i, j]$, which stores the edit distance between two strings $x[1..i]$ and $y[1..j]$ for $(i, j) \in \{0, \dots, |x|\} \times \{0, \dots, |y|\}$.

Example 2.9 Figure 2.1 shows the edit graph for two strings “ACTC” and “ACGT” after computing EDIT-DISTANCE (Figure 2.1(a)) and TRACEBACK (Figure 2.1(b)), where all the edit costs are assumed to be 1 (unit cost), i.e.

$$d(a, b) = \begin{cases} 0 & \text{if } a = b, \\ 1 & \text{otherwise,} \end{cases}$$

for any $(a, b) \in (\Sigma \cup \{-\} \times \Sigma \cup \{-\}) \setminus \{-, -\}$. In Figure 2.1(a), the distance (minimum weight) from top left to bottom right in the edit graph is computed, and it turns out to be 2 as given at the bottom right node. Figure 2.1(b) shows two possible optimal alignments (or edit scripts). Since, by the procedure TRACEBACK, replacements are preferred to deletions and insertions, the former alignment is obtained by this procedure.

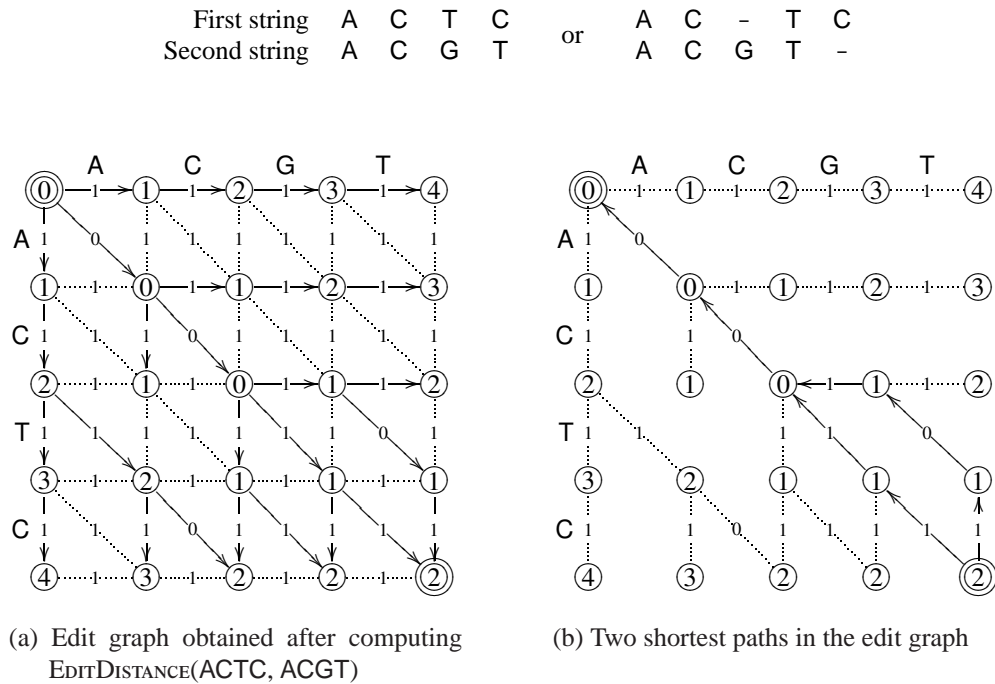


Figure 2.1. Dynamic programming algorithm for string edit distance

Improvements of Dynamic Programming Based Algorithms

Algorithms for string edit distance based on dynamic programming basically run in quadratic time, i.e. for two strings of size n in $\Theta(n^2)$ time, and it is computationally expensive for long strings. To improve the computational complexity, many attempts have been made, and some sub-quadratic time algorithms have been proposed based on the *Four-Russians* technique [MP80], bit-parallelism [Mye99], LZ78 compression [CLZU02], and so forth.

A *fixed-parameter* algorithm is also proposed (cf. [Gus97, Section 12.2]) by fixing an upper bound k of the number of applying edit operations. This algorithm runs in $O(kn)$ time. Bodlaender *et al.* [BDFW95] investigated the *parameterized complexity* of the LCS problem for multiple strings.

2.2.6 Declarative Definition

Here we consider an alternative definition of string edit distance by using the notion of *trace* [WF74]. A trace is an order-preserving mapping between two strings.

Definition 2.10 (Trace [WF74]) For two strings $x = a_1 \cdots a_m$ and $y = b_1 \cdots b_n$, a *trace* of x and y is $M \subseteq \{1, \dots, m\} \times \{1, \dots, n\}$ satisfying the following conditions.

1. $\forall (i, j), (i', j') \in M [i = i' \iff j = j']$,
2. $\forall (i, j), (i', j') \in M [i < i' \iff j < j']$.

Note that these two conditions are integrated into one condition: $\forall (i, j), (i', j') \in M [i \leq j \iff i' \leq j']$. For two strings $x = a_1 \cdots a_m$ and $y = b_1 \cdots b_n$, we denote

$$M_D = \{1, \dots, m\} \setminus \{i \in \{1, \dots, m\} \mid \exists j (i, j) \in M\},$$

$$M_I = \{1, \dots, n\} \setminus \{j \in \{1, \dots, n\} \mid \exists i (i, j) \in M\}.$$

We define the cost of trace between two strings x and y as follows:

$$\text{cost}(M) = \sum_{(i,j) \in M} d(a_i, b_j) + \sum_{i \in M_D} d(a_i, -) + \sum_{j \in M_I} d(-, b_j).$$

Intuitively, the elements of M_D indicate the indices of symbols to be deleted from x , and the elements of M_I indicate the indices of symbols to be deleted from y (or to be inserted into x).

Let us denote the set of all possible traces between x and y by $\mathcal{M}(x, y)$. Wagner and Fisher showed that the string edit problem is reduced into the optimization problem of traces.

Theorem 2.11 (Theorem 1 in [WF74]) For two string x and y ,

$$D^{\text{Edit}}(x, y) = \min_{E \in \mathcal{E}(x, y)} \text{cost}(E) = \min_{M \in \mathcal{M}(x, y)} \text{cost}(M).$$

We refer to the definition describing *what* string edit distance is by using the notion of trace as the *declarative definition* of it.

Example 2.12 A trace of “ACTC” and “ACGT” is shown in **Figure 2.2(a)**, while Figure 2.2(b) is not a trace since it violates the condition of trace in Definition 2.10. ■

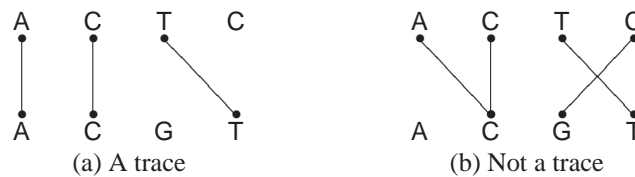


Figure 2.2. Example of a trace

Since both the edit and alignment problems can be reduced into the same combinatorial optimization problem of traces in Theorem 2.11, these two problems are computationally equivalent, i.e. for any two strings x and y , the following holds:

$$\mathbf{D}^{\text{EDIT}}(x, y) = \mathbf{D}^{\text{ALN}}(x, y).$$

By using the notion of trace, Kececioglu *et al.* formalized the alignment problem for more than two strings (multiple alignment) as a combinatorial optimization problem with integer linear programming [KLM⁺00].

In **Table 2.1**, we summarize three operational definitions of string edit distance. In this table, all the edit operations are applied only to source strings to transform them to a target string. All these definitions can be reduced into a declarative definition by trace.

Table 2.1. Three views of string edit distance

Problem	Source(s)	Target	Operations		
			DEL	INS	REP
Edit	x	y	✓	✓	✓
Approximate common subsequence	x, y	z	✓		✓
Approximate common supersequence (Alignment)	x, y	z		✓	✓

x, y : two input strings, z : a common string obtained by editing x and y
DEL, INS, and REP stand for deletion, insertion, and replacement operations respectively.

2.2.7 Approximation of String Edit Distance

If we want a set of strings similar to a given string pattern among a large data set of strings, it is required to speed up the computation of string edit distance even by sacrificing the accuracy of the computation.

In addition, the metric space of string edit distance is not tractable. Actually, for a set of strings X , the metric space is represented as a set of pairwise distances between strings in X with an $n \times n$ table. Then, it is difficult to see a comprehensive structure of these strings in the metric space. If this metric space can be embedded into a more familiar and tractable metric space such as Euclidean space while preserving the distances between each pair of strings, it may gain the understanding of the whole structure of given data.

Driven by these motivations, a variety of filtering (cf. [NBYST01]) and embedding (cf. [Cor03]) methods have been proposed.

Filtering by a Lower Bound

Ukkonen introduced a distance measure between two strings called *q-gram distance* [Ukk92], which gives a lower bound for string edit distance (this is also known as a *non-expanding embedding*). The basic idea of *q-gram distance* is simple: the more the different substrings occur between two strings, the more distant these are. A string (text) analysis based on *q-gram* dates from Shannon's paper [Sha48]. A *q-gram* (or *n-gram*[†]) is a string in Σ^q for $q \in \mathbb{N}$.

For two strings $x, w \in \Sigma^*$, if there exist $y, z \in \Sigma^*$ such that $x = y \cdot w \cdot z$, then x has an *occurrence* of w . Let $\#x[w]$ denote the total number of occurrences of w in x , i.e.

$$\#x[w] = |\{y \mid x = y \cdot w \cdot z \wedge y, z \in \Sigma^*\}|.$$

The *q-gram profile* of a string x is the vector $G_q(x) = (\#x[w])_{w \in \Sigma^q}$, indexed by all *q-grams* w and arranged in lexicographic order of *q-grams*.

Definition 2.13 (q-Gram Distance) For two strings x and y in Σ^* , and $q \in \mathbb{N}$, the *q-gram distance* between x and y is defined by

$$\mathbf{D}_q^{\text{GRAM}}(x, y) = \|G_q(x) - G_q(y)\|_1 = \sum_{w \in \Sigma^q} |\#x[w] - \#y[w]|.$$

[†]These “ q ” and “ n ” are no more than parameter symbols. Thus, it may also be mentioned as *k-gram*, *l-gram*, and so on.

Example 2.14 For $\Sigma = \{a, b\}$, consider two strings $x = abaaa$ and $y = bbaaaa$ in Σ^* . The 2-gram profiles are $G(x)_2 = (2, 1, 1, 0)$ and $G(y)_2 = (3, 0, 1, 1)$.

w	aa	ab	ba	bb
$\#x[w]$	2	1	1	0
$\#y[w]$	3	0	1	1

Therefore, the 2-gram distance between x and y is $D_2^{\text{GRAM}}(x, y) = 3$. ■

Note that q -gram distance is not a metric but a *pseudometric*.

Example 2.15 For $\Sigma = \{a, b\}$, consider two strings $x = abaa$ and $y = baab$ in Σ^* . The 2-gram profiles of x and y are the same: $G(x)_2 = G(y)_2 = (1, 1, 1, 0)$. Then the 2-gram distance between x and y is $D_2^{\text{GRAM}}(x, y) = 0$ in spite of $x \neq y$. ■

Theorem 2.16 (from Theorem 5.1 in [Ukk92]) For two strings x and y , and a natural number $q \in \mathbb{N}$, the following holds.

$$\frac{D_q^{\text{GRAM}}(x, y)}{2q} \leq D_1^{\text{EDIT}}(x, y).$$

The q -gram distance $D_q^{\text{GRAM}}(x, y)$ can be evaluated in time $O(|x| + |y|)$ and in space $O(|\Sigma|^q + |x| + |y|)$ [Ukk92]. By Theorem 2.16, we can compute a lower bound of unit-cost edit distance much faster than the edit distance itself, and this property is applied to efficient filtering for string search based on unit-cost edit distance.

For a set of strings $X = \{x_1, x_2, \dots, x_n\}$ ($x_i \in \Sigma^*$ for $i \in \{1, \dots, n\}$), and a string pattern $p \in \Sigma^*$, let us denote all the strings in X within unit-cost edit distance k from p by

$$X_{\text{EDIT}}^{\leq k}(p) = \{x \in X \mid (\exists x \in X) D_1^{\text{EDIT}}(p, x) \leq k\},$$

and all the strings in X within q -gram distance k from p by

$$X_{\text{GRAM}}^{\leq k}(p) = \{x \in X \mid (\exists x \in X) D_q^{\text{GRAM}}(p, x) \leq k\}.$$

When we want the set $X_{\text{EDIT}}^{\leq k}(p)$, we can narrow this set by first computing $X_{\text{GRAM}}^{\leq 2qk}(p)$, since by Theorem 2.16 the following holds.

$$X_{\text{EDIT}}^{\leq k}(p) \subseteq X_{\text{GRAM}}^{\leq 2qk}(p).$$

Following q -gram based filtering, many filtering methods have been proposed such as the wavelet-based method by Kahveci and Singh [KS01], the spaced seed method by Ma *et al.* [MTL02] and the gapped q -gram method by Burkhardt and Kärkkäinen [BK03].

String Edit Distance Embeddings

Attempts to embed a set of pairwise distances into a more tractable metric space such as a low-dimensional Euclidean space have a long history. For example, Metric multi-dimensional scaling (MMDS) (cf. [KWU06]) is a well-known method mainly for visualization. Most of algorithms for MDS have been developed based on heuristics without theoretical quality assurance of embeddings.

Recently, theoretical studies of embeddings have revealed some remarkable facts. Now let us denote by ℓ_1 normed spaces (with arbitrary dimension). If there exists a mapping $\phi : \Sigma^n \rightarrow \ell_1$ such that for any two string x and y ,

$$D_1^{\text{EDIT}}(x, y) \leq \|\phi(x) - \phi(y)\|_1 \leq \delta \cdot D_1^{\text{EDIT}}(x, y),$$

then we say that the metric space of unit-cost edit distance can be *embeddable* into ℓ_1 spaces with δ -distortion ($\delta \geq 1$). Many attempts have been made for achieving a lower distortion δ . For strings over $\{0, 1\}^n$, Krauthgamer and Rabani [KR06] showed a lower bound of distortion $\Omega(\log n)$, and Ostrovsky and Rabani [OR05] showed an upper bound of distortion $2^{O(\sqrt{\log n \log \log n})}$ with a probabilistic polynomial time algorithm. The dimensionality of embedded spaces due to Ostrovsky and Rabani [OR05] is, however, at least quadratic in n . Thus, it is difficult to develop efficient algorithms straightforward from the embedding. (Note that these results on strings over $\{0, 1\}^n$ is extendable to larger alphabets.)

$\dagger 2^{O(\sqrt{\log n \log \log n})} = o(n^c)$ for any constant $c > 0$.

From a pragmatic point of view, it is very important to develop an efficient embedding algorithm with the best possible distortion (cf. [Bad06]). By using a dimensionality-reduction technique, Batu *et al.* addressed this problem, and proposed an efficient algorithm that approximates unit-cost edit distance within a factor of nearly $\delta \approx n^{1/3}$ in almost linear time [BES06].

Cormode and Muthukrishnan addressed the approximation problem of string edit distance with an additional edit operations — *substring move* (or *block move*) operations. Shapira and Storer [SS02] showed that the string edit problem with substring move operations is NP-complete by reducing the BIN-PACKING problem into this edit problem. Interestingly, in approximation of string edit distance, the addition of *move operations* makes the problem easier as opposed to exact computation. This contrast is attributed to the fact that approximation algorithms do not constructively compute edit scripts or traces. Cormode and Muthukrishnan proposed an efficient algorithm that approximates unit-cost edit distance with substring move operations within a factor of $\delta = O(\log \log^* n)^\dagger$ in almost linear time [CM07, CM02, Cor03].

2.3 Basic Notation for Trees

Trees we consider in this thesis are mainly labeled rooted trees, in which each node is labeled from a finite alphabet. An *ordered tree* is a tree in which the left-to-right order among siblings is given. An *unordered tree* is a tree with no order among siblings. In order to formulate these trees, we employ a subclass of partially ordered set theory (or lattice theory) and its algebraic system rather than graph theory since approximate pattern matching between two trees is considered as an order-preserving mapping between two ordered sets.

2.3.1 Rooted Trees

We define a tree as a subclass of a partially ordered set. A *partially ordered set* (or a *poset* for short) is a set V with a binary relation \leq (called a *partial order*), denoted by (V, \leq) , that satisfies the following:

1. $\forall x \in V \quad (x \leq x)$ (reflectivity),
2. $\forall x, y \in V \quad (x \leq y \wedge y \leq x \implies x = y)$ (antisymmetry),
3. $\forall x, y, z \in V \quad (x \leq y \wedge y \leq z \implies x \leq z)$ (transitivity).

If the set V in a poset (V, \leq) is finite, we say the poset is *finite*. Two elements $x, y \in V$ do not always satisfy either $x \leq y$ or $y \leq x$. Thus, if $x, y \in V$ satisfy either $x \leq y$ or $y \leq x$, two elements x and y are said to be *comparable*. In contrast, if x and y is not comparable, x and y are said to be *incomparable*. We write $x < y$ if $x \leq y$ and $x \neq y$. Also, we often write $y \geq x$ and $y > x$ for $x \leq y$ and $x < y$ respectively.

Let (V, \leq) be a poset, and U be a nonempty subset of V . A node $x \in U$ is *minimal* in U if, for all $y \in U$ such that $y \leq x$, it holds that $x = y$. The node x is called *minimum* if x is a unique minimal nodes. If any two elements of V are comparable, then we refer to (V, \leq) as a *chain* or a *totally ordered set*, and to \leq as a *linear order* or a *total order*.

Definition 2.17 (Rooted Trees) A *rooted tree* T is a non-empty finite poset (V, \leq) that satisfies the following:

1. There exists a unique element $r \in V$ such that $x \leq r$ for all $x \in V$,
2. For all $x, y, z \in V$, if $x \leq y$ and $x \leq z$, then y and z are comparable.

The elements of V are called *nodes* (or *vertices*) of T , and the node r is called the *root* of T and denoted by $\text{root}(T)$.

We refer to the binary relation \leq as the *hierarchical order*, where, for two nodes $x \leq y$, we say that x is an *ancestor* of y , and y is a *descendent* of x . Also, for two nodes $x < y$, we say that x is a *proper ancestor* of y , and y is a *proper descendent* of x .

For a tree T , and a node $x \in T$, by $(\uparrow x)_T$ (resp. $(\downarrow x)_T$) we denote the set of all ancestors (resp. proper ancestors) of x in T , i.e.

$$(\uparrow x)_T = \{y \in T \mid x \leq y\}, \quad (\downarrow x)_T = \{y \in T \mid x < y\}.$$

[†] $\log^* n$ is called *iterated logarithm* of n , and is the number of application times of \log function to get a constant, i.e. $\log^* n = 0$ if $n \leq 1$; otherwise $\log^* n = 1 + \log^*(\log n)$.

Note that for any $x \in T$, the sets $(\uparrow x)_T$ and $(\downarrow x)_T$ form chains, and Definition 2.17.2 can be replaced with the following equivalent condition:

- 2'. The set $(\uparrow x)_T$ is a chain for every $x \in V$.

For a tree T , by $V(T)$, we denote the set of all nodes in T , and by \leq_T the hierarchical order \leq of T for clarity. We also write $x \in T$ instead of $x \in V(T)$ for short.

The *parent* of a non-root node x , denoted by $\text{par}(x)$, is the minimum node y in the set $\{z \in V \mid z > x\}$, and conversely, the node x is called a *child* of $\text{par}(x)$. The set of all children of a node x is denoted by $\text{ch}(x)$, i.e. $\text{ch}(x) = \{y \in V \setminus \{\text{root}(T)\} \mid \text{par}(y) = x\}$. For any two distinct children of a node, one node is said to be a *sibling* of the other. A node with no children is called a *leaf*. The set of all leaves in a tree T is denoted by $\text{leaves}(T)$. The *depth* of a node x is, denoted by $\text{dep}(x)$, the number of proper ancestors of x , i.e. $\text{dep}(x) = |\{y \mid x < y\}|$. The depth of any root node is 0. By $\text{dep}(T)$ we denote the maximum depth of T , i.e. $\text{dep}(T) = \max\{\text{dep}(x) \mid x \in T\}$, and call it the *depth* or *height* of T . The size of a tree T is the number of nodes in T , denoted by $|T|$. For a node x , the size of $\text{ch}(x)$ is denoted by $\text{deg}(x)$, and referred to as the *degree* of x . The maximum number of children for all nodes in a tree T is denoted by $\text{deg}(T)$, i.e. $\text{deg}(T) = \max\{\text{deg}(x) \mid x \in T\}$, and referred to as the *degree* of T .

Note that a rooted tree T pursuant to Definition 2.17 is naturally regarded as a directed graph. In fact, by defining the set of nodes as $V(T)$ and the set of directed edges as $E(T) = \{(x, \text{par}(x)) \mid x \in V(T) \setminus \{\text{root}(T)\}\}$, $(V(T), E(T))$, we have a directed graph $G = (E(T), V(T))$.

A tree may be equipped with another order in addition to the hierarchical order. This additional order is called the *sibling order*, denoted by \preceq , and defines the left-to-right relation between nodes.

Definition 2.18 (Rooted Ordered Trees) A *rooted ordered tree* T is a triplet (V, \leq, \preceq) such that the pair (V, \leq) is a rooted tree, and the pair (V, \preceq) is a non-empty finite poset that satisfies the following:

1. For any $x, y \in V$, two nodes x and y are comparable with respect to the sibling order if and only if x and y are equivalent, or incomparable with respect to the hierarchical order.
2. For any distinct nodes $x, y, x', y' \in V$, if $x \leq x', y \leq y'$ and $x' \preceq y'$, then $x \preceq y$.

We define the notation \prec, \succ , and \succeq for the sibling order in the same way as the hierarchical order. For two nodes $x \prec y$, we say that x is to the *left* of y , and y is to the *right* of x . Also we denote by \preceq_T the sibling order in a tree T .

We refer to the rooted trees without the sibling order as the *unordered trees*, and to the rooted ordered trees as the *ordered trees* for short. Also, we use the term *trees* simply to refer to both the ordered and unordered trees if there is no confusion. For example, if we say ‘‘This property holds for a tree,’’ we mean that the property holds no matter whether the relevant tree is ordered or unordered. By $\mathcal{T}, \mathcal{T}_U$, and \mathcal{T}_O , we denote the set of all trees, unordered trees, and ordered trees with finite nodes respectively.

We define a *forest* by omitting the condition 1 in Definition 2.17. As in the case of trees, we define *ordered forests* and *unordered forests*, and use the term *forests* to refer to both ordered and unordered forests.

Definition 2.19 (Forests) An *unordered forest* F is a finite poset (V, \leq) that satisfies the following:

1. For all $x, y, z \in V$, if $x \leq y$ and $x \leq z$, then y and z are comparable.

An *ordered forest* F is a triplet (V, \leq, \preceq) such that the pair (V, \leq) is an unordered forest, and the pair (V, \preceq) is a poset that satisfies the following:

2. For any $x, y \in V$, two nodes x and y are comparable with respect to the sibling order if and only if x and y are equivalent, or incomparable with respect to the hierarchical order.
3. For any distinct nodes $x, y, x', y' \in V$, if $x \leq x', y \leq y'$ and $x' \preceq y'$, then $x \preceq y$.

As in the case of trees, for a forest $F = (V, \leq)$, we denote by $V(F)$ the set of nodes V , and by \leq_F the hierarchical order \leq , and by \preceq_F the sibling order for clarity. We also write $x \in F$ instead of $x \in V(F)$ for short. The size of a forest F is the number of nodes in F , denoted by $|F|$. By $\mathcal{F}, \mathcal{F}_U$, and \mathcal{F}_O , we denote the set of all forests, unordered forests, and ordered forests with finite nodes respectively.

Definition 2.20 (Forest Induced by a Set of Nodes) Let T be a tree, and U be a subset of $V(T)$. A forest of T induced by a set of nodes U is a forest $T[U] = (U, \leq)$ (or (U, \leq, \preceq) for ordered trees) defined as follows:

1. $V(T[U]) = U$,
2. $\forall x, y \in U [x \leq_{T[U]} y \iff x \leq_T y]$,
3. $\forall x, y \in U [x \preceq_{T[U]} y \iff x \preceq_T y]$ (only for ordered trees).

By $F(v)$ we denote the forest of T induced by $V(T(v)) \setminus \{v\}$.

Example 2.21 Figure 2.3 depicts a tree T , and a forest of T induced by U , i.e. $T[U]$, where

$$U = \{t_2, t_3, t_5, t_7, t_8, t_{10}, t_{12}, t_{13}, t_{15}\}.$$

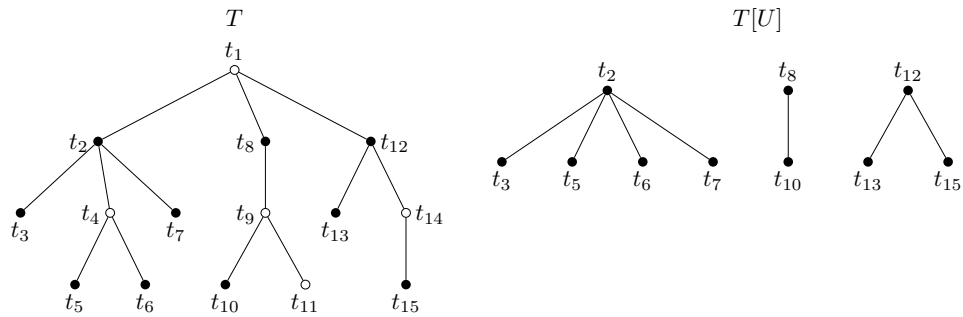


Figure 2.3. A tree T and a forest of T induced by U

Normally, trees and forests are denoted with upper-case letters and nodes with lower-case letters. Preferably, we use for trees letters R, S, T ; for forests the letter F . Optionally, subscripts and primes are used.

A forest F' is called a *subforest* of a forest F if $V(F')$ is a subset of $V(F)$, and the hierarchical order (and the sibling order for an ordered tree) of F' are inherited from F .

Definition 2.22 (Complete Subtree) Let T be a tree. A *complete subtree* of T rooted at $v \in T$ is a tree S defined by

1. $V(S) = \{x \in T \mid x \leq_T v\}$,
2. $\forall x, y \in S [x \leq_S y \iff x \leq_T y]$,
3. $\forall x, y \in S [x \preceq_S y \iff x \preceq_T y]$ (only for ordered trees).

By $T(v)$ we denote the complete subtree of T rooted at $v \in T$. In the same manner, we define a *complete subtree* of a forest F rooted at $v \in F$.

A forest F' is called a *complete subforest* of a forest F if F' consists of complete subtrees in F .

Remark 2.1 The term *subtree* is used in several meanings. In this definition, by $T(v)$ we denote the complete subtree rooted at $v \in T$. On the other hand, for any subset of nodes $U \in V(T)$ such that $T[U]$ forms a tree, if every edge in $T[U]$ is also an edge in T , then the tree $T[U]$ is referred to as the *subtree* of T , otherwise, a *subtree pattern* of T .

Definition 2.23 (Labeled Trees and Labeled Forests) Let Σ be a nonempty finite set of symbols, called *alphabet*, and let $l : V \rightarrow \Sigma$ be a *labeling function* from a set of nodes V to an alphabet Σ . If all the nodes in a tree or a forest are labeled by a labeling function l , we say that the tree or the forest is *labeled*.

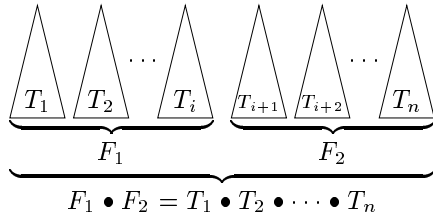


Figure 2.4. The composite of two forests F_1 and F_2

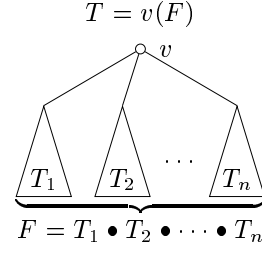


Figure 2.5. Tree $T = v(F)$

We use sans-serif typeface for labels, e.g. $\Sigma = \{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \dots\}$.

We introduce the notion of the *least common ancestor* (also known as the *lowest or nearest common ancestor*) of a set of nodes. This notion plays a significant role in comparing structures of two trees.

Definition 2.24 (Least Common Ancestor) For a tree T and a set of nodes $U \subseteq V(T)$, a *common ancestor* of U is a node $x \in T$ such that $y \leq x$ for all $y \in U$. The *least common ancestor* of U is the common ancestor x of U such that $x \leq y$ holds for any common ancestor y of U , and denoted by $\text{lca}(U)$.

For a tree T , when the set of nodes $U \subseteq V(T)$ includes just two nodes x and y , we have a function $\smile: V \times V \rightarrow V$ defined by

$$x \smile y = \text{lca}(\{x, y\}).$$

Example 2.25 For the tree T in Figure 2.3, we can observe the following.

$$\begin{aligned} \text{lca}(\{t_3, t_4, t_6\}) &= t_2, & \text{lca}(\{t_3, t_6, t_9\}) &= t_1, & \text{lca}(\{t_3\}) &= t_3, \\ t_3 \smile t_6 &= t_2, & t_4 \smile t_9 &= t_1, & t_8 \smile t_{10} &= t_8. \end{aligned}$$

2.3.2 Syntactic Representation of Ordered Trees and Forests

An ordered forest is viewed as a series of subtrees T_1, \dots, T_n as shown in **Figure 2.4**, and a new tree is constructed by adding a new node so that it is the parent of the roots of all subtrees as shown in **Figure 2.5**. From this observation, we introduce yet another representation of ordered trees and forests in a syntactical way. This syntactical representation is inspired by the notation due to Dulucq and Touzet [DT03b].

Definition 2.26 (Syntactic Representation of Ordered Trees and Forests) Let T_1, \dots, T_n ($n \geq 0$) be a series of ordered trees. By $T_1 \bullet \dots \bullet T_n$ we denote a concatenation of ordered trees T_1, \dots, T_n in order. We refer to a concatenation of ordered trees as an *ordered forest*. If $n = 0$, i.e. an ordered forest with no trees, we refer to it as an *empty forest*, and simply write \emptyset . We also use the notation $\bullet_{i=1}^n T_i$ instead of $T_1 \bullet \dots \bullet T_n$ for short. Let us define $T_i \bullet \dots \bullet T_j = \emptyset$ for $i > j$.

Let F be an *ordered forest* $T_1 \bullet \dots \bullet T_n$ and v be a node not included in $V(F)$. The ordered tree $v(F)$ is defined as follows.

- $V(v(F)) = (\bigcup_{i=1}^n V(T_i)) \cup \{v\}$.
- $x < y$ if and only if one of the following holds.
 - $x, y \in T_i$ for some i , and $x < y$ in T_i .
 - $y = v$.
- $x \prec y$ if and only if one of the following holds.
 - $x, y \in T_i$ for some i , and $x \prec y$ in T_i .
 - $x \in T_i, y \in T_j$ and $i < j$.

Algorithm 2.2 Tree traversals

<pre> procedure PREORDER($v(F) \bullet F'$) visit(v) PREORDER(F) if $F \neq \emptyset$ PREORDER(F') if $F' \neq \emptyset$ end </pre>	<pre> procedure POSTORDER($v(F) \bullet F'$) POSTORDER(F) if $F \neq \emptyset$ POSTORDER(F') if $F' \neq \emptyset$ visit(v) end </pre>
--	---

This representation can be defined in recursive form, and it is convenient for decomposing a tree structure recursively. In the following, we give the definition in a mutually recursive manner (See Figure 2.4 and Figure 2.5).

Definition 2.27 (Recursive Definition of Ordered Trees and Forests) Let \mathcal{T}_0 be the set of *ordered trees* defined hereinafter, and let \bullet be a connective symbol. The set of *ordered forests* is the smallest set \mathcal{F}_0 such that:

- (F1) $T \in \mathcal{T}_0 \implies T \in \mathcal{F}_0$.
- (F2) $T_1, \dots, T_n \in \mathcal{T}_0 \implies T_1 \bullet \dots \bullet T_n \in \mathcal{F}_0$.
- (F3) $F_1, \dots, F_n \in \mathcal{F}_0 \implies F_1 \bullet \dots \bullet F_n \in \mathcal{F}_0$.

Let \mathcal{V} be the set of nodes. The set of *ordered trees* is the smallest set \mathcal{T}_0 such that:

- (T1) $v \in \mathcal{V} \implies v \in \mathcal{T}_0$.
- (T2) $v \in \mathcal{V} \wedge F \in \mathcal{F}_0 \implies v(F) \in \mathcal{T}_0$.

For an ordered forest $F = T_1 \bullet \dots \bullet T_n$, we write $T_i \in F$ for any $i \in \{1, \dots, n\}$.

Example 2.28 The tree in Figure 2.3 is represented as

$$t_1(t_2(t_3 \bullet t_4(t_5 \bullet t_6) \bullet t_7) \bullet t_8(t_9(t_{10} \bullet t_{11})) \bullet t_{12}(t_{13} \bullet t_{14}(t_{15}))).$$

Note that a forest can be \emptyset , whereas a tree must include at least one node. A tree and a forest are usually represented by T and F (possibly with a subscript or a superscript), respectively. In comparing two forests or trees, we use the following matching conventions:

$$\begin{aligned}
T \bullet F = T_1 \bullet \dots \bullet T_n &\implies T = T_1 \wedge F = T_2 \bullet \dots \bullet T_n, \\
F \bullet T = T_1 \bullet \dots \bullet T_n &\implies T = T_n \wedge F = T_1 \bullet \dots \bullet T_{n-1}, \\
T \bullet F = T' &\implies T = T' \wedge F = \emptyset, \\
F \bullet T = T' &\implies T = T' \wedge F = \emptyset, \\
v = v'(F) &\implies v = v' \wedge F = \emptyset.
\end{aligned}$$

2.3.3 Tree Traversals

A tree traversal is a way of enumerating all the nodes in trees. Here, we present two basic traversal schemes for ordered trees, *preorder* and *postorder* traversals. In a *left-to-right* preorder traversal, the root of a tree is first visited, and then the subtrees rooted at its children are visited from left to right recursively. (In a *right-to-left* preorder traversal, these children are visited from right to left recursively.)

On the other hand, in a *left-to-right* postorder traversal, the root of a tree is visited after all the subtrees rooted at its children are visited from left to right recursively. (In a *right-to-left* postorder traversal, these children are visited from right to left recursively.) We refer to *left-to-right* preorder or postorder simply as preorder or postorder if otherwise stated.

In **Algorithm 2.2**, we show the procedures for the left-to-right preorder and postorder traversals. These procedures are initially called as $\text{PREORDER}(T)$ and $\text{POSTORDER}(T)$ for an ordered tree T . The procedure $\text{visit}(v)$ depends on the application. We have the procedures for right-to-left preorder and postorder traversals by rewriting the first line of each procedure respectively as follows:

```

procedure PREORDER( $F' \bullet v(F)$ )
procedure POSTORDER( $F' \bullet v(F)$ )

```


Example 2.29 Consider a labeled ordered tree T in **Figure 2.6**, in which each label is attached to the left of each node. Figure 2.6(a) depicts preorder numbering of T , in which each number is attached to the right of each node, and Figure 2.6(b) depicts postorder numbering of T . According to these orders, we can serialize the labels in T . We refer to these serialized labels as *label sequences* of T .

The label sequence of T in left-to-right preorder is “abcdef” while the label sequence of T in left-to-right postorder is “cedbfa.”

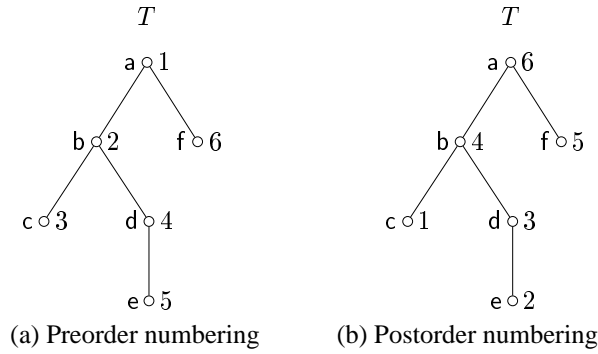


Figure 2.6. Left-to-Right Preorder and Postorder numberings of an ordered tree

For a tree T , by \preceq_T we denote the left-to-right preorder in T . Then, the formal definition is given as follows.

Definition 2.30 (Left-to-Right Preorder) For a tree T , the *left-to-right preorder* of T is the minimum total order \preceq_T satisfying the following: for any $x, y \in V(T)$,

- $x \leq_T y \implies x \preceq_T y$,
- $x \preceq_T y \implies x \preceq_T y$,

If $x \neq y$ and $x \preceq_T y$, we denote $x \triangleleft_T y$. In other words, for a tree T , the left-to-right preorder of T is a common linear extension[†] of the hierarchical order and sibling order of T . It is easy to see that the left-to-right preorder of T is uniquely determined, and totally ordered on $V(T)$.

2.4 Tree Edit Distance — Tai Distance

Tree edit distance is a generalization of string edit distance. In this section, we review the operational definition of tree edit distance along the lines of string edit distance.

2.4.1 Edit Operations

The tree edit distance between two trees is defined as the minimum cost of elementary edit operations required to transform one tree into the other [Tai79, ZS89].

The set of elementary *edit operations* on a tree T consists of the following three operations.

Replacement of a node x in T by a new node y not in T .

Deletion of a non-root node x from T , moving all children of x right under the parent of x .

Insertion of a new node x into T as a child of a node y in T , moving a subset (a consecutive subsequence in the case of ordered trees) of y 's children and their descendants right under the new node x . Note that this is the complementary operation of deletion.

[†]A totally ordered set (X, \trianglelefteq) is a *linear extension* of a poset (X, \leq) if for any $x, y \in X$, $x \leq y \implies x \trianglelefteq y$ holds.

Remark 2.2 (Root-Editable Operations) According to the definition of edit operations, we cannot delete the root and insert a new root of any tree. This setting is just for theoretical tractability. Actually, in most algorithms of (general) tree edit distance, it is allowed to edit the root of any tree as well as the other nodes, and edit operations are defined on forests instead of trees. We may assume, without loss of generality, that the root of any tree remains intact by any edit operation since we can add an abiding dummy root on the top of the root of the original tree, and regard it as the new root. By removing the dummy root after applying all the edit operations, we have the same effect of root-editable operations. In fact, in spite of the definition of edit operations, we employ root-editable operations in this thesis.

As in the case of strings, in order to estimate the cost for edit operations to transform a tree S to a tree T , we denote each *edit signature* as follows:

- “ $s \mapsto t$ ” for the replacement of a node s in S by a node t in T ,
- “ $\varepsilon \mapsto t$ ” for the insertion of a node t in T into S ,
- “ $s \mapsto \varepsilon$ ” for the deletion of a node s from S .

The symbol ε denotes a *null node*, and we assume the label of any null node ε is a *gap symbol*, i.e. $l(\varepsilon) = “-.”$

Let us abuse notation by referring to each edit operation as its edit signature although the edit signatures do not have enough information for actual edit operations if there is no confusion. We write $S \xrightarrow{e} T$, if we obtain a tree T from a tree S by applying an edit operation e ,

Let d be a cost function of edit signatures, and we equate the two notations $d(x \mapsto y)$ and $d(x, y)$ as follows:

$$d(s \mapsto t) = d(s, t) \quad \text{for all } (s, t) \in (V(S) \cup \{\varepsilon\}) \times (V(T) \cup \{\varepsilon\}).$$

Note that each edit signature is used just for a cost factor of each edit operation. Then an edit signature cannot be a representation of edit operation itself. In fact, the edit signature for insertion does not have any information about where the node is to be inserted. Moreover, almost all conventional models based on edit distance have factored in just a transition of the labels of edited nodes, because this simplification enables us to estimate the cost for edit operations regardless of the applicative order. It may oversimplify in a specific application, and establishing a more general edit model is an open problem to address, although it goes beyond the scope of this thesis.

Example 2.31 Figure 2.7 shows that the three elementary edit operations. ■

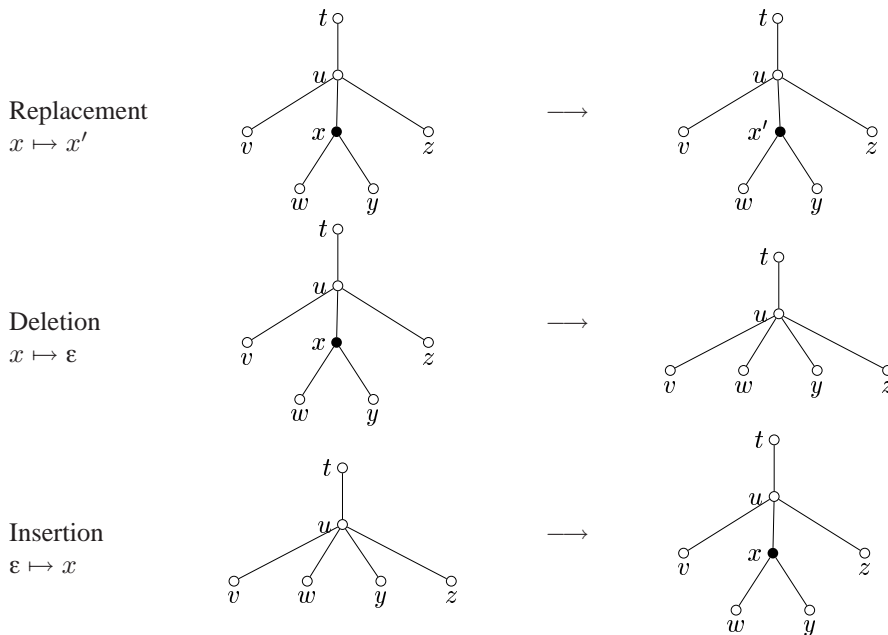


Figure 2.7. Examples of the three elementary edit operations

2.4.2 Tree Edit Problem

We here forbid edit operations with the edit signatures $x \mapsto y$ such that $x = y$. If a series of edit operations $E = \langle e_1, \dots, e_n \rangle$ ($n \geq 1$) transforms a tree S into a tree T , there exists a series of trees $\langle T_0, \dots, T_n \rangle$ such that $T_0 = S$, $T_n = T$, and the i -th edit operation $e_i = (s_i \mapsto t_i)$ transforms T_{i-1} into T_i for $i \in \{1, \dots, n\}$, i.e.

$$S = T_0 \xrightarrow{e_1} T_1 \xrightarrow{e_2} \dots \xrightarrow{e_n} T_n = T$$

If $S = T$, then we define $n = 0$. The cost function $\text{cost}(\cdot)$ for a series of edit operations $E = \langle e_1, \dots, e_n \rangle$ is derived from the total cost for the elementary edit operations as follows:

$$\text{cost}(E) = \sum_{i=1}^n d(e_i).$$

If $n = 0$, then we define $\text{cost}(E) = 0$. We refer to the series of edit operations E as the *edit script*. By $\mathcal{E}(S, T)$ we denote the set of all possible edit scripts to transform S into T .

Tai presented the following edit distance for trees [Tai79]. Although a variety of tree edit distance measures have been proposed, Tai's distance measure is recognized as the most standard one, and we refer to it as *Tai distance* to distinguish it from the other variants of tree edit distance.

Definition 2.32 (Tai Distance [Tai79]) The edit distance between two trees S and T is defined as follows:

$$\mathbf{D}^{\text{Tai}}(S, T) = \min_{E \in \mathcal{E}(S, T)} \text{cost}(E).$$

We refer to an edit script with the minimum cost as an *optimal edit script*. The problem of computing the edit distance between two trees along with an optimal edit script is called the *tree edit problem*, or more specifically *Tai edit problem*.

Tai showed that if d is a metric, then the tree edit distance is also a metric. In particular, Tai defined the cost function d as a metric over node labels, i.e.

$$d(x, y) = d_l(l(x), l(y)) \quad \text{for all } (x, y) \in (V(S) \cup \{\varepsilon\}) \times (V(T) \cup \{\varepsilon\}), \quad (2.1)$$

where $d_l : (\Sigma \cup \{-\}) \times (\Sigma \cup \{-\}) \rightarrow \mathbb{R}$ is a metric. Throughout this thesis, we assume that the cost function d is a metric.

We refer to Definition 2.32 as an *operational definition* of tree edit distance, or an *operational semantics* of tree edit distance. We refer to a specific algorithm for computing tree edit distance also as an operational definition (or semantics).

2.4.3 Algorithms for Ordered Trees

The algorithms for computing Tai distance for ordered trees have been continuously improved since Tai proposed an $O(n^6)$ time and space algorithm [Tai79] in 1979, where n denotes the size of inputs ($n = \max\{|T_1|, |T_2|\}$ for given two trees T_1 and T_2). Subsequently, Aoki presented a top-down algorithm [Aok83] with the time and space complexity of $O(n^4)$, and it was followed by a bottom-up algorithm [Tan83] which runs in $O(n^4)$ time and $O(n^3)$ space. Zhang and Shasha independently developed a bottom-up algorithm [ZS89] which runs in $O(n^4)$ time and $O(n^2)$ space, and Klein improved their algorithm, and proposed an algorithm [Kle98] with $O(n^3 \log n)$ time and $O(n^3 \log n)$ space (this space bound can be improved to $O(n^2)$ due to Klein's comments cited in a survey [Bil05, Section 3.2.3]). It is notable that Klein's algorithm runs in the same computational complexity even for *unrooted* ordered trees.

These algorithms are all based on decomposing tree structures with dynamic programming, and have been improved by refining *decomposition strategy* for trees. Dulueq and Touzet conducted an intensive analysis of decomposition strategy for trees [DT03b, DT05], and proved a worst-case lower bound of $\Omega(n^2 \log^2 n)$ time for any decomposition strategy based algorithms [DT05, Corollary 16].

Later, without dynamic programming, Chen proposed a novel algorithm [Che01] by reducing the tree edit problem into a matrix multiplication problem. Chen's algorithm runs in $O(n^{3.5})$ time and $O(n^3)$ space. Then, Klein's algorithm had been still asymptotically most efficient.

Table 2.2. Computational complexity of Tai edit problem for ordered trees

Reference	Specific complexity		n -Parameterized	
	Time	Space	Time	Space
[Tai79]	$O(n_1 n_2 h_1^2 h_2^2)$	$O(n_1 n_2 h_1^2 h_2^2)$	$O(n^6)$	$O(n^6)$
[Aok83]	$O(n_1 n_2 h_1 h_2)$	$O(n_1 n_2 h_1 h_2)$	$O(n^4)$	$O(n^4)$
[Tan83]	$O(n_1^2 n_2 \ell_1)$	$O(n_1 n_2 \ell_1)$	$O(n^4)$	$O(n^3)$
[ZS89]	$O(n_1 n_2 c_1 c_2)$	$O(n_1 n_2)$	$O(n^4)$	$O(n^2)$
[Kle98]	$O(n_1^2 n_2 \log n_2)$	$O(n_1 n_2)^\dagger$	$O(n^3 \log n)$	$O(n^2)$
[Che01]	$O(n_1 n_2 + \ell_1^2 n_2 + \ell_1^{2.5} \ell_2)$	$O((n_1 + \ell_1^2) c_2 + n_2)$	$O(n^{3.5})$	$O(n^3)$
[DMRW07] [‡]	$O(n_1^2 n_2 (1 + \log \frac{n_2}{n_1}))$	$O(n_1 n_2)$	$O(n^3)$	$O(n^2)$
Worst-case lower bounds in decomposition strategy based algorithms				
[DT03b, DT05]	$\Omega(n_1 n_2 \log n_1 \log n_2)$		$\Omega(n^2 \log^2 n)$	
[DMRW07]	$\Omega(n_1 n_2^2 (1 + \log \frac{n_1}{n_2}))$		$\Omega(n^3)$	
Fixed-parameter algorithms			Remark	
[SZ90]	$O(K^2 n_1 \min\{\ell_1, \ell_2\})$	$O(n_1 n_2)$	unit costs	
[Tou05]	$O(k^3 n)$	$O(kn)$	$O(k^2 n)$ time for traceback	

n_i : size of tree T_i for $i \in \{1, 2\}$, and assume that $n_1 \leq n_2 = n$,

h_i : height (depth) of tree T_i for $i \in \{1, 2\}$, $\ell_i = |\text{leaves}(T_i)|$ for $i \in \{1, 2\}$,

$c_i = \min\{\ell_i, h_i\}$ for $i \in \{1, 2\}$, K : fixed upper bound of distance.

k : fixed upper bound of the number of insertions and deletions.

[†] This bound is reported in a survey [Bil05, Section 3.2.3] as Klein's comments.

[‡] A worst-case optimal algorithm in decomposition strategy based algorithms.

Based on the decomposition strategy framework proposed by Dulueq and Touzet [DT03b, DT05], the most recent improvement has been made by Demaine *et al.* [DMRW07]. They presented an $O(n^3)$ -time and $O(n^2)$ -space algorithm and tightened the worst-case lower bound of $\Omega(n^2 \log^2 n)$ time by Dulueq and Touzet [DT05] to $\Omega(n^3)$ time. Thus, their algorithm is known to be worst-case optimal in decomposition strategy.

Shasha and Zhang proposed a *fixed-parameter algorithm* [SZ90] for unit-cost Tai distance. This algorithm runs in $O(K^2 \cdot \min\{n_1, n_2\} \cdot \ell)$ time and $O(n_1 n_2)$ space for a fixed upper bound K of unit-cost Tai distance, where $\ell = \min\{|\text{leaves}(T_1)|, |\text{leaves}(T_2)|\}$. This algorithm returns the unit-cost Tai distance between given two trees if the distance is at most K , otherwise stops. Touzet proposed a *fixed-parameter algorithm* [Tou05] without unit-cost restriction. This algorithm runs in $O(k^3 n)$ time and $O(kn)$ space for a fixed upper bound k of the number of insertions and deletions.

We summarize the complexities of these algorithms in **Table 2.2**. Among these algorithms, in what follows, we show Zhang and Shasha's algorithm since it is succinct and is the basis of the other decomposition-based algorithms.

Zhang-Shasha's Algorithm for Ordered Trees

Zhang and Shasha's operational definition of Tai distance is simple and almost the same as the definition of string edit distance. In their definition, Tai distance is defined on two forests instead of two trees.

$$\begin{aligned}
 D(\emptyset, \emptyset) &= 0 \\
 D(F \bullet v(F'), \emptyset) &= D(F \bullet F', \emptyset) + d(v, \varepsilon) \\
 D(\emptyset, F \bullet v(F')) &= D(\emptyset, F \bullet F') + d(\varepsilon, v) \\
 D(F_1 \bullet v_1(F'_1), F_2 \bullet v_2(F'_2)) &= \min \begin{cases} D(F_1, F_2) + D(F'_1, F'_2) + d(v_1, v_2) \\ D(F_1 \bullet F'_1, F_2 \bullet v_2(F'_2)) + d(v_1, \varepsilon) \\ D(F_1 \bullet v_1(F'_1), F_2 \bullet F'_2) + d(\varepsilon, v_2) \end{cases} \quad (2.2)
 \end{aligned}$$

Algorithm 2.3 Zhang-Shasha's algorithm for Tai distanceInput: T_1, T_2

```

compute  $ll(\cdot)$ ,  $\text{keyroots\_index}(T_1)$ ,  $\text{keyroots\_index}(T_2)$ 
foreach  $m \in \text{keyroots\_index}(T_1)$  in ascending order do
  foreach  $n \in \text{keyroots\_index}(T_2)$  in ascending order do
     $\text{TREEDIST}(m, n)$ 
return  $D_T[|T_1|, |T_2|]$ 

```

procedure $\text{TREEDIST}(m, n)$

```

 $D(\emptyset, \emptyset) \leftarrow 0$ 
for  $i \leftarrow ll(m)$  to  $m$ 
   $D(T_1[ll(m)..i], \emptyset) \leftarrow D(T_1[ll(m)..i-1], \emptyset) + d(T_1[i], \epsilon)$ 
for  $j \leftarrow ll(n)$  to  $n$ 
   $D(\emptyset, T_2[ll(n)..j]) \leftarrow D(\emptyset, T_2[ll(n)..j-1]) + d(\epsilon, T_2[j])$ 
for  $i \leftarrow ll(m)$  to  $m$ 
  for  $j \leftarrow ll(n)$  to  $n$ 
    if  $ll(i) = ll(m)$  and  $ll(j) = ll(n)$  then
       $D(T_1[ll(m)..i], T_2[ll(n)..j]) \leftarrow$ 
      
$$\min \begin{cases} D(T_1[ll(m)..i-1], T_2[ll(n)..j]) + d(T_1[i], \epsilon) \\ D(T_1[ll(m)..i], T_2[ll(n)..j-1]) + d(\epsilon, T_2[j]) \\ D(T_1[ll(m)..i-1], T_2[ll(n)..j-1]) + d(T_1[i], T_2[j]) \end{cases}$$

       $D_T[i, j] \leftarrow D(T_1[ll(m)..i], T_2[ll(n)..j])$ 
    else
       $D(T_1[ll(m)..i], T_2[ll(n)..j]) \leftarrow$ 
      
$$\min \begin{cases} D(T_1[ll(m)..i-1], T_2[ll(n)..j]) + d(T_1[i], \epsilon) \\ D(T_1[ll(m)..i], T_2[ll(n)..j-1]) + d(\epsilon, T_2[j]) \\ D(T_1[ll(m)..i-1], T_2[ll(n)..j-1]) + D_T[i, j] \end{cases}$$

  end
end

```

The decomposition strategy of this algorithm is simple, i.e. it always focuses on the rightmost roots of two forests. Any complete subforest F'_1 of F_1 (resp. F'_2 of F_2) occurs in the computation $D(F_1, F_2)$ as an argument is called a *relevant forest* of F_1 (resp. F_2). It is easy to see that the number of relevant forests dominates the computational complexity of this type of decomposition-based algorithms.

In order to compute these recurrences efficiently, Eq.(2.2) is split into the following two recurrences.

$$D(F_1 \bullet v_1(F'_1), F_2 \bullet v_2(F'_2)) = \min \begin{cases} D(F_1, F_2) + D_T(v_1(F'_1), v_2(F'_2)) \\ D(F_1 \bullet F'_1, F_2 \bullet v_2(F'_2)) + d(v_1, \epsilon) \\ D(F_1 \bullet v_1(F'_1), F_2 \bullet F'_2) + d(\epsilon, v_2) \end{cases}$$

$$D_T(v_1(F_1), v_2(F_2)) = \min \begin{cases} D(F_1, F_2) + d(v_1, v_2) \\ D(F_1, v_2(F_2)) + d(v_1, \epsilon) \\ D(v_1(F_1), F_2) + d(\epsilon, v_2) \end{cases}$$

where $D(F_1, F_2)$ denotes the Tai distance between two forests F_1 and F_2 , and $D_T(T_1, T_2)$ denotes the Tai distance between two trees T_1 and T_2 . As shown in **Algorithm 2.3**, Zhang and Shasha implemented the recurrences efficiently with dynamic programming. In this algorithm, all the nodes in each tree T_i ($i \in \{1, 2\}$) are indexed by left-to-right postorder numbering from 1 to $|T_i|$. For a tree T , by $T[i]$ we denote the node indexed by i , and by $T[i..j]$ we denote a forest induced by the set of nodes $T[i], T[i+1], \dots, T[j]$. If $i > j$, then $T[i..j] = \emptyset$. By $ll(i)$ we denote the index of the leftmost leaf in the subtree rooted at $T[i]$. The array $D_T[i, j]$ stores the Tai distance between two trees $T_1(T_1[i])$ and $T_2(T_2[j])$, and $D(F_1, F_2)$ denotes an abstract array (or a hash table) which stores the Tai distance between two forests F_1 and F_2 .

The essential idea of this dynamic programming algorithm is left-to-right postorder numbering of nodes. In the numbering, a set of nodes indexed by consecutive numbers induces a forest, and the forest $T[l(i)..i]$ forms the complete subtree rooted at $T[i]$ for any $i \in \{1, \dots, |T|\}$. By virtue of such properties, this numbering enables right-to-left decomposition of forests in a bottom-up manner.

The set of nodes $\text{keyroots}(T)$ is defined as follows:

$$\text{keyroots}(T) = \{\text{root}(T)\} \cup \{v \in T \mid v \text{ has a left sibling}\}.$$

In implementation, the indices of the nodes in $\text{keyroots}(T)$ are computed as $\text{keyroots_index}(T)$.

$$\text{keyroots_index}(T) = \{k \in \{1, \dots, |T|\} \mid \nexists k' \in \{1, \dots, |T|\} \text{ such that } k < k' \text{ and } ll(k) = ll(k')\}.$$

Example 2.33 For a given tree T in **Figure 2.8(a)**, the nodes in $\text{keyroots}(T)$ are depicted by filled circles:

$$\text{keyroots_index}(T) = \{3, 5, 8, 10, 11, 13, 15, 17, 18, 19\}.$$

Each iteration in the procedure $\text{TREEDIST}(m, n)$ is computed along each thick line from a leaf to an ancestor depicted by a filled circle. Then, the thick lines and isolated filled circles in **Figure 2.8(a)** depict the decomposition strategy of Zhang-Shasha’s algorithm for T (cf. [DT05]). In this strategy, a given tree is decomposed along leftmost disjoint paths as shown in **Figure 2.8(a)**. We can observe that $T[ll(16)..16]$ forms the complete subtree rooted at $T[16]$ since $ll(16) = 12$. **Figure 2.8(b)** shows the forest induced by the set of nodes $\{T[i] \mid 7 \leq i \leq 17\}$, i.e. $T[7..17]$. ■

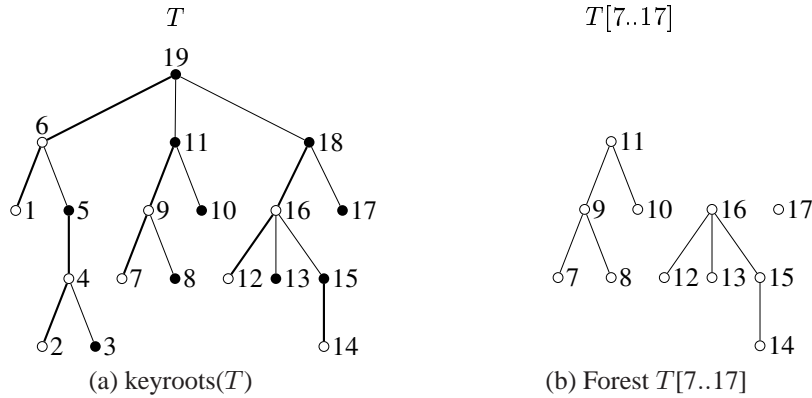


Figure 2.8. Left-to-right postorder numbering of nodes

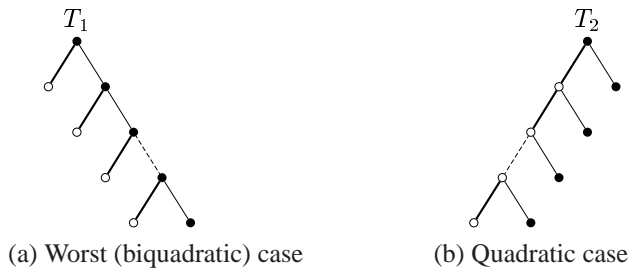


Figure 2.9. Two extreme examples in Zhang-Shasha’s algorithm

Complexity. We here estimate the time complexity of Zhang-Shasha’s algorithm. For a tree T and a node $v \in T$, we define the *collapsed depth* of v as the number of ancestors of v included in $\text{keyroots}(T)$, and denote it by $\text{cdepth}(v)$, i.e.

$$\text{cdepth}(v) = |(\uparrow v)_T \cap \text{keyroots}(T)|.$$

Also, let $\text{cdepth}(T)$ denote

$$\text{cdepth}(T) = \max\{\text{cdepth}(v) \mid v \in T\}.$$

It follows from the recurrence in Eq.(2.2) that the number of relevant forests for each node $v \in \text{keyroot}(T)$ is $|T(v)| - 1$. Then, we obtain the upper bound of the number of relevant forests for a tree T as follows:

$$\sum_{v \in \text{keyroots}(T)} |F(v)| < \sum_{v \in \text{keyroots}(T)} |T(v)| = \sum_{v \in T} \text{cdepth}(v) \leq \sum_{v \in T} \text{cdepth}(T) = |T| \cdot \text{cdepth}(T).$$

Zhang and Shasha showed that $\text{cdepth}(T) \leq \min\{\text{dep}(T), \text{leaves}(T)\}$ for a tree T in [ZS89, Lemma 6]. Hence, Zhang-Shasha's algorithm runs in $O(|T_1| \cdot |T_2| \min\{\text{dep}(T_1), \text{leaves}(T_1)\} \min\{\text{dep}(T_2), \text{leaves}(T_2)\})$.

Dulucq and Tichit [DT03a] conducted an exact complexity analysis of Zhang-Shasha's algorithm [ZS89], and showed that the average time complexity of the algorithm is $\Theta(n^3)$. The worst case of $\Theta(n^4)$ time happens if the same two trees T_1 and T_1 in **Figure 2.9** are given [DT03a], while the time complexity is $\Theta(n^2)$ if two trees T_2 and T_2 are given. In Figure 2.9, the filled circles indicate the elements of $\text{keyroots}(T_i)$ for $i \in \{1, 2\}$, and we assume $|T_1| = |T_2| = n$.

Decomposition-based Algorithms

Following Zhang-Shasha's algorithm [ZS89], two remarkable algorithms were proposed: Klein's algorithm [Kle98] and an optimal decomposition algorithm due to Demaine *et al.* [DMRW07]. These three algorithms are all based on the decomposition strategy.

Definition 2.34 (Decomposition Strategy [DT03b, Definition 3]) For two ordered forests F_1 and F_2 , consider Tai distance between F_1 and F_2 . Let

$$\begin{aligned} F_1 &= T_1^L \bullet F_1^R = F_1^L \bullet T_1^R, \\ F_2 &= T_2^L \bullet F_2^R = F_2^L \bullet T_2^R, \end{aligned}$$

where T_i^L and T_i^R are trees, and F_i^L and F_i^R are forests for $i \in \{1, 2\}$. A decomposition of F_i is a *left-decomposition* if F_i is decomposed into T_i^L and F_i^R for each $i \in \{1, 2\}$. A decomposition of F_i is a *right-decomposition* if F_i is decomposed into F_i^L and T_i^R for each $i \in \{1, 2\}$.

A *decomposition strategy* is denoted by a mapping $\mathcal{S} : \mathcal{F}_O \times \mathcal{F}_O \rightarrow \{\text{left}, \text{right}\}$. We refer to left or right as a *direction*. In strategy, the direction indicates the way of decomposition for each pair of forests.

Example 2.35 In Eq.(2.2) in Zhang-Shasha's algorithm, each forest is decomposed into the *rightmost* tree and the rest of forest, i.e. a forest $F = T_1 \bullet \dots \bullet T_{n-1} \bullet T_n$ is decomposed into $T_1 \bullet \dots \bullet T_{n-1}$ and T_n . This strategy is depicted by $\mathcal{S}(F_1, F_2) = \text{right}$ for any forests F_1 and F_2 . ■

The decomposition strategies for Zhang-Shasha's algorithm and Klein's algorithm are depicted as follows.

- **Zhang-Shasha's algorithm [ZS89]:** $\mathcal{S}(F_1, F_2) = \text{right}$.
- **Klein's algorithm [Kle98]:** Assume that $|F_1| \geq |F_2|$, then

$$\mathcal{S}(F_1, F_2) = \begin{cases} \text{left} & \text{if } |F_1^L| \leq |F_1^R| \\ \text{right} & \text{otherwise.} \end{cases}$$

In the decomposition strategy in Klein's algorithm, for given two forests F_1 and F_1 (assume $|F_1| \geq |F_2|$), the number of decomposed forests is bound by $O(|F_1| \log |F_1|)$ and $O(|F_2|^2)$ respectively. Since these upper bounds determine the time complexity of Klein's algorithm, it runs in $O(|T_1| \cdot |T_2|^2 \log |T_1|)$ for given two trees T_1 and T_2 (assume $|T_1| \geq |T_2|$).

Klein's algorithm determines the direction of decomposition according only to the complete subforests of F_1 even if $|F_1'| \leq |F_2'|$ holds for a pair of relevant forests F_1' and F_2' in a subproblem, where F_1' and F_2' are complete subforests of F_1 and F_2 respectively. It is a legitimate question to ask if Klein's decomposition strategy can be applied also to a subforest of F_2 if $|F_1'| \leq |F_2'|$ holds in a subproblem.

Demaine *et al.* tackled this problem, and designed a new succinct algorithm, and proved that it is an optimal algorithm in decomposition-based algorithms [DMRW07].

Approximate Tree Matching with Variable-Length Don't Care Patterns

In approximate string matching, a variable-length-don't-care pattern (VLDC pattern) is an element of $(\Sigma \cup \{*\})^*$, where the symbol “*” is a VLDC symbol, which matches any substring with cost 0 [Aku96]. For example a VLDC pattern “a*ndment” matches a string “alignment” with distance 1 since “*” matches “lig” with cost 0 and “d” is deleted with cost 1 (we assume unit costs).

Zhang *et al.* extended the notion of VLDC pattern matching in strings to trees, and proposed an algorithm for approximate tree matching between two ordered trees with VLDC patterns [ZSW94] as a variant of the Tai edit problem. In the algorithm, the following two VLDC symbols were introduced: (1) a path-VLDC, which matches part of a path from the root to a leaf of a tree; (2) an umbrella-LDC, which matches part of a path and all the subtrees ramifying from the nodes, except at the lowest node of the path. This algorithm runs in the same computational complexity of Zhang-Shasha's algorithm for Tai distance.

2.4.4 Algorithms for Unordered Trees

For unordered trees, Zhang proposed an algorithm [ZSS92] for computing Tai distance for unordered trees with a similar dynamic programming procedure as Algorithm 2.3, and showed it runs in $O(n_1 n_2 + \ell_1! 3^{\ell_1} (\ell_1^3 + d_2^2) n_2)$ time, where by d_2 we denote $\deg(T_2)$, and the other parameters are the same as in Table 2.2.

Also Zhang *et al.* [ZSS92] showed that the decision problem of determining whether the Tai distance between two given trees T_1 and T_2 is less than or equal to $k \in \mathbb{N}$ (*Tai distance problem*) is NP-complete, even for binary trees with an alphabet of size two, by reducing *Exact Cover by 3-Sets (X3C)* (cf. [GJ79, page 221]) into this decision problem. Moreover, Zhang and Jiang [ZJ94] showed that the largest common subtree problem and the Tai edit problem is shown to be MAX SNP-hard, also even for binary trees with an alphabet of size two, by reducing *Maximum Bounded Covering by 3-Sets (MAX 3SC-3)* [Kan91] into this optimization problem. This fact implies that the Tai edit problem does not have any polynomial-time approximation scheme (PTAS), unless P=NP, where a problem has a PTAS if the problem can be approximated within a factor of $1 + \epsilon$ (for any constant $\epsilon > 0$) in polynomial time.

Torsello *et al.* proposed an algorithm [TH03a] by reducing the problem of computing Tai distance for unordered trees into the *maximum weighted clique problem* in order to take advantage of a powerful heuristics for approximation [BPS00]. Horesh *et al.* developed an A* algorithm [HMU06] for *unlabeled* unordered trees.

We summarize the computational complexity of the Tai edit problem for unordered trees in **Table 2.3** and the algorithms for Tai distance for unordered trees in **Table 2.4**.

Table 2.3. Computational complexity of Tai edit problem for unordered trees

Reference	Worst-case lower bound	Remark
[ZSS92]	NP-complete	even for binary trees with an alphabet of size two
[ZJ94]	MAX SNP-hard	''

Table 2.4. Algorithms for Tai distance for unordered trees

Reference	Method	Remark
[ZSS92]	dynamic programming	$O(n_1 n_2 + \ell_1! 3^{\ell_1} (\ell_1^3 + d_2^2) n_2)$ time
[TH03a]	reduction to maximum weighted clique problem	approximation algorithm
[HMU06]	A* algorithm	for unlabeled trees

n_i : size of tree T_i for $i \in \{1, 2\}$, $\ell_i = |\text{leaves}(T_i)|$ for $i \in \{1, 2\}$

2.5 Tree Mappings

Tai showed a fundamental correspondence between the effect of a series of edit operations and a common pattern shared in two trees [Tai79]. The common pattern is represented as a set of pairs of nodes called a *tree mapping* (originally called just a *mapping* [Tai79]). Tai distance and the other variants are defined by giving a condition of tree mappings as well as the operational ways. We refer to this static view of tree edit distance by a class of tree mappings as the *declarative definition* or as the *declarative semantics*.

Basically, a declarative definition of a tree edit distance measure makes it clearer and easier to study mathematical properties of the measure. In fact, most algorithms for computing various tree edit distance measures have been designed from the view of tree mappings, and the correctness of each algorithm also has been proved mainly by verifying the correspondence between the algorithm and the declarative definition of the edit distance measure.

In what follows, we first introduce a general form of tree mapping, and next show the declarative definition of Tai distance by using tree mapping.

2.5.1 Tree Mapping based Distance

A tree mapping is a partial node-to-node correspondence between two trees.

Definition 2.36 (Tree Mapping) A *tree mapping* M from a tree S to a tree T is a set of pairs of nodes $M \subseteq V(S) \times V(T)$ satisfying $s_1 = s_2 \Leftrightarrow t_1 = t_2$ for any $(s_1, t_1), (s_2, t_2) \in M$.

We also say that a tree mapping *between* S and T if there is no confusion. We denote by $\mathcal{M}(S, T)$ all possible tree mappings between S and T , and by M_{ST} we sometimes denote an element of $\mathcal{M}(S, T)$.

For a tree mapping M between two trees S and T , we use the following notation:

$$\begin{aligned} M(s) &= t \text{ if } \exists t \in T [(s, t) \in M] \text{ for } s \in S, \\ M^{-1}(t) &= s \text{ if } \exists s \in S [(s, t) \in M] \text{ for } t \in T, \\ M^{(1)} &= \{s \in S \mid \exists t \in T [(s, t) \in M]\}, \\ M^{(2)} &= \{t \in T \mid \exists s \in S [(s, t) \in M]\}. \end{aligned}$$

Given a cost function $d : (V(S) \cup \{\varepsilon\}) \times (V(T) \cup \{\varepsilon\}) \rightarrow \mathbb{R}$ as shown in Eq.(2.1), the cost of a tree mapping M is defined as follows:

$$\text{cost}(M) = \sum_{(s,t) \in M} d(s, t) + \sum_{s \in V(S) \setminus M^{(1)}} d(s, \varepsilon) + \sum_{t \in V(T) \setminus M^{(2)}} d(\varepsilon, t). \quad (2.3)$$

In this chapter, we introduce a variety of classes of tree mappings by imposing some restriction on tree mappings, and define tree edit distance measures such as Tai distance, alignment distance and constrained distance based on these classes of tree mappings. These classes are referred to as the symbol \mathcal{C} when we need a general description for each class of tree mapping, and we refer to the tree mapping \mathcal{C} -mapping. Let $\mathcal{M}^{\mathcal{C}}(S, T)$ denote the set of all possible tree mappings belonging to class \mathcal{C} between two trees S and T , i.e.

$$\mathcal{M}^{\mathcal{C}}(S, T) = \{M \in \mathcal{M}(S, T) \mid M \text{ is a tree mapping from } S \text{ to } T \text{ belonging to class } \mathcal{C}\}.$$

In what follows, we give a few important properties of tree mappings related to the notion of tree mapping class.

Definition 2.37 (Class Hierarchy of Tree Mappings) A class \mathcal{C}_1 of tree mappings is a *subclass* of a class \mathcal{C}_2 of tree mappings, denoted by $\mathcal{C}_1 \subseteq \mathcal{C}_2$, if $\mathcal{M}^{\mathcal{C}_1}(S, T) \subseteq \mathcal{M}^{\mathcal{C}_2}(S, T)$ holds for any two tree S and T . Conversely, we say that \mathcal{C}_2 is a *superclass* of \mathcal{C}_1 if $\mathcal{C}_1 \subseteq \mathcal{C}_2$. In particular, \mathcal{C}_1 is a *proper subclass* of a class \mathcal{C}_2 , denoted by $\mathcal{C}_1 \subsetneq \mathcal{C}_2$, if $\mathcal{C}_1 \subseteq \mathcal{C}_2$ and $\mathcal{M}^{\mathcal{C}_1}(S, T) \subsetneq \mathcal{M}^{\mathcal{C}_2}(S, T)$ for some two trees S and T hold. Conversely, we say that \mathcal{C}_2 is a *proper superclass* of \mathcal{C}_1 if $\mathcal{C}_1 \subsetneq \mathcal{C}_2$.

Definition 2.38 (Monotonicity of a Class of Tree Mappings) A class \mathcal{C} of tree mappings is *monotonic* if the following holds:

$$(\forall S, T \in \mathcal{T})(\forall M, M' \in \mathcal{M}(S, T)) [M' \subseteq M \wedge M \in \mathcal{M}^{\mathcal{C}}(S, T) \implies M' \in \mathcal{M}^{\mathcal{C}}(S, T)].$$

Definition 2.39 (Composition of Tree Mappings) Let R , S , and T be trees. For any two tree mappings $M_{RS} \in \mathcal{M}(R, S)$ and $M_{ST} \in \mathcal{M}(S, T)$, the *composition* of M_{RS} and M_{ST} is defined as follows:

$$M_{ST} \circ M_{RS}^{\dagger} = \{ (r, t) \in V(R) \times V(T) \mid \exists s \in S [(r, s) \in M_{RS} \wedge (s, t) \in M_{ST}] \}.$$

Lemma 2.40 (Subadditivity of Tree Mapping Costs) Let R , S , and T be trees. If the cost function d is a metric, then for any two tree mappings $M_{RS} \in \mathcal{M}(R, S)$ and $M_{ST} \in \mathcal{M}(S, T)$, the following holds.

$$\text{cost}(M_{ST} \circ M_{RS}) \leq \text{cost}(M_{RS}) + \text{cost}(M_{ST}).$$

Proof. For any $(r_1, t_1), (r_2, t_2) \in M_{23} \circ M_{12}$, the condition $r_1 = r_2 \Leftrightarrow t_1 = t_2$ holds due to Definition 2.36. Then, we consider the cases for the nodes of each tree.

1. For any node $s \in S$, it suffices to consider the following four cases:
 - (a) $s \in M_{RS}^{(2)}$ and $s \in M_{ST}^{(1)}$,
 - (b) $s \notin M_{RS}^{(2)}$ and $s \in M_{ST}^{(1)}$,
 - (c) $s \in M_{RS}^{(2)}$ and $s \notin M_{ST}^{(1)}$,
 - (d) $s \notin M_{RS}^{(2)}$ and $s \notin M_{ST}^{(1)}$.
 In any case, there exist two unique node pairs $(r, s) \in (V(R) \cup \{\varepsilon\}) \times V(S)$ and $(s, t) \in V(S) \times (V(T) \cup \{\varepsilon\})$ associated with s . Since $d(r, t) \leq d(r, s) + d(s, t)$, the assertion holds in these cases.
2. For any node $r \in R$, if $r \in M_{RS}^{(1)}$, then there exists a unique node pair $(r, s) \in M_{RS}$. Hence, it is considered in the previous cases 1(a) and 1(c). Otherwise, $r \notin M_{RS}^{(1)}$ holds, and $d(r, \varepsilon)$ is considered both in $\text{cost}(M_{RS} \circ M_{ST})$ and $\text{cost}(M_{RS})$. This factor $d(r, \varepsilon)$ does not make any difference between $\text{cost}(M_{ST} \circ M_{RS})$ and $\text{cost}(M_{RS}) + \text{cost}(M_{ST})$.
3. For any node $t \in T$, by symmetry, it is similar to the previous case.

Therefore, the assertion holds in any cases. ■

We define two important properties of a class of tree mappings.

Definition 2.41 (Symmetricity of a Class of Tree Mappings) A class \mathcal{C} of tree mappings is *symmetric* if $\mathcal{M}^{\mathcal{C}}(S, T) = \mathcal{M}^{\mathcal{C}}(T, S)$ holds for any two trees S and T .

Definition 2.42 (Transitivity of a Class of Tree Mappings) Let R , S , and T be arbitrary trees. A class \mathcal{C} of tree mappings is *transitive* if, for any two tree mappings $M_{RS} \in \mathcal{M}^{\mathcal{C}}(R, S)$ and $M_{ST} \in \mathcal{M}^{\mathcal{C}}(S, T)$, the composite $M_{ST} \circ M_{RS}$ is a tree mapping such that $M_{RT} \in \mathcal{M}^{\mathcal{C}}(R, T)$.

We introduce tree mapping based distance according to the class of tree mappings.

Definition 2.43 (\mathcal{C} -Distance) Given a class \mathcal{C} of tree mappings, the \mathcal{C} -distance between two trees S and T is defined as follows:

$$\mathbf{D}^{\mathcal{C}}(S, T) = \min_{M \in \mathcal{M}^{\mathcal{C}}(S, T)} \text{cost}(M).$$

We call this definition a *declarative definition* of \mathcal{C} -distance.

If a \mathcal{C} -mapping M between two trees has the minimum cost, we call M an *optimal \mathcal{C} -mapping*. In practical use, we often normalize distance as follows:

$$\mathbf{ND}^{\mathcal{C}}(S, T) = \frac{\mathbf{D}^{\mathcal{C}}(S, T)}{\max\{|S|, |T|\}}.$$

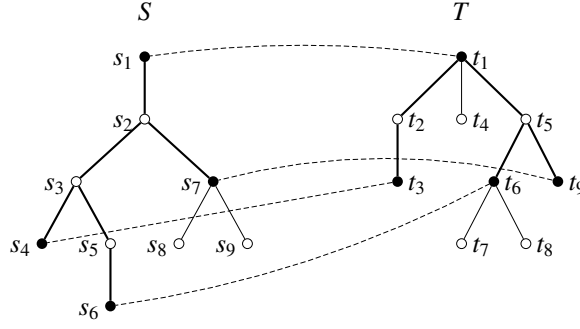


Figure 2.10. Tai mapping

Proposition 2.44 (Lower/Upper Bound of \mathcal{C} -Distance) Given two classes \mathcal{C}_1 and \mathcal{C}_2 of tree mappings, if \mathcal{C}_1 is a subclass of \mathcal{C}_2 , then \mathcal{C}_1 -distance is an upper bound of \mathcal{C}_2 -distance, i.e.

$$\mathcal{C}_1 \subseteq \mathcal{C}_2 \implies \forall S, T \in \mathcal{T} [\mathbf{D}^{\mathcal{C}_2}(S, T) \leq \mathbf{D}^{\mathcal{C}_1}(S, T)].$$

Proof. It is obvious from Definition 2.37. ■

Proposition 2.45 (Transitivity of \mathcal{C} -Distance)

If a class \mathcal{C} of tree mappings is transitive, then \mathcal{C} -distance is transitive.

Proof. Recall that we assume that the cost function d is a metric. Let R , S , and T be trees. Since \mathcal{C} is transitive, $M_{ST} \circ M_{RS}$ belongs to class \mathcal{C} of tree mappings for any $M_{RS} \in \mathcal{M}^{\mathcal{C}}(R, S)$, $M_{ST} \in \mathcal{M}^{\mathcal{C}}(S, T)$, and $M_{RT} \in \mathcal{M}^{\mathcal{C}}(R, T)$. Since \mathcal{C} is transitive, it holds that $M_{ST} \circ M_{RS} \in \mathcal{M}^{\mathcal{C}}$. Without loss of generality, we may assume that M_{RS} , M_{ST} , and M_{RT} are tree mappings with minimum costs. By Lemma 2.40 and the definition of \mathcal{C} -distance, we have

$$\mathbf{D}^{\mathcal{C}}(R, T) = \text{cost}(M_{RT}) \leq \text{cost}(M_{ST} \circ M_{RS}) \leq \text{cost}(M_{RS}) + \text{cost}(M_{ST}) = \mathbf{D}^{\mathcal{C}}(R, S) + \mathbf{D}^{\mathcal{C}}(S, T).$$
■

2.5.2 Tai Mapping — Declarative Definition of Tai Distance

Tai proposed the following class of tree mappings for reducing the problem of computing the minimum cost of edit scripts into the problem of computing the minimum cost of tree mappings [Tai79]. Tai formulated tree edit distance in accordance with the paper by Wagner and Fisher [WF74] for string edit distance.

Definition 2.46 (Tai Mapping [Tai79]) A tree mapping $M \subseteq V(S) \times V(T)$ is said to be a *Tai mapping* if the following are satisfied for any $(s_1, t_1), (s_2, t_2) \in M$.

1. $s_1 = s_2 \iff t_1 = t_2$.
2. $s_1 < s_2 \iff t_1 < t_2$.
3. $s_1 \prec s_2 \iff t_1 \prec t_2$ (only for ordered trees).

Note that the first and second conditions are replaced with $s_1 \leq s_2 \iff t_1 \leq t_2$.

This class is indicated by $\mathcal{C} = \text{Tai}$. Then, by $\mathcal{M}^{\text{Tai}}(S, T)$, we refer to the set of Tai mappings between two trees S and T .

Example 2.47 For two ordered trees S and T in **Figure 2.10**, $M = \{(s_1, t_1), (s_4, t_3), (s_6, t_6), (s_7, t_9)\}$ is a Tai mapping from S to T . The Tai mapping is depicted by dashed lines. For two ordered trees S and T in **Figure 2.11**, $M = \{(s_1, t_2), (s_2, t_3), (s_3, t_4)\}$ is not a Tai mapping since $t_4 \leq t_2$ does not hold although $s_3 \leq s_1$ holds. ■

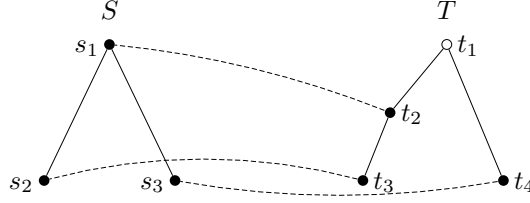


Figure 2.11. Non-Tai mapping

Proposition 2.48 (Transitivity of Tai Mapping, Lemma 3.1(1) in [Tai79]) Tai mapping is transitive.

Proof. Let R , S and T be trees. Consider two Tai mappings $M_{RS} \in \mathcal{M}^{\text{Tai}}(R, S)$ and $M_{ST} \in \mathcal{M}^{\text{Tai}}(S, T)$. For any $(r_1, t_1), (r_2, t_2) \in M_{ST} \circ M_{RS}$, there exist $s_1, s_2 \in S$ such that $(r_1, s_1), (r_2, s_2) \in M_{RS}$ and $(s_1, t_1), (s_2, t_2) \in M_{ST}$. By the definition of Tai mappings, the following hold:

1. $r_1 = r_2 \Leftrightarrow s_1 = s_2$ and $s_1 = s_2 \Leftrightarrow t_1 = t_2$,
2. $r_1 \leq_R r_2 \Leftrightarrow s_1 \leq_S s_2$ and $s_1 \leq_S s_2 \Leftrightarrow t_1 \leq_T t_2$,
3. $r_1 \preceq_R r_2 \Leftrightarrow s_1 \preceq_S s_2$ and $s_1 \preceq_S s_2 \Leftrightarrow t_1 \preceq_T t_2$ (only for ordered trees).

Therefore, $M_{ST} \circ M_{RS}$ is also a Tai mapping from R to T . ■

In the following lemma, we show an important correspondence between edit scripts and Tai mappings.

Lemma 2.49 (Edit Script and Tai Mapping) Let S and T be two trees. The following two properties hold between the costs of edit scripts and Tai mappings.

1. For any Tai mapping $M \in \mathcal{M}(S, T)$, there exists an edit script $E \in \mathcal{E}(S, T)$ such that $\text{cost}(E) = \text{cost}(M)$.
2. For any edit script $E = \langle e_1, \dots, e_n \rangle \in \mathcal{E}(S, T)$, there exists a Tai mapping $M \in \mathcal{M}(S, T)$ such that $\text{cost}(E) \leq \text{cost}(M)$.

Proof. Without loss of generality, for any Tai mapping $M \in \mathcal{M}(S, T)$, we assume $(\text{root}(S), \text{root}(T)) \in M$ (cf. Remark 2.2).

1. From the definition of the cost of a tree mapping in Eq.(2.3), we can construct a corresponding edit script E with the same cost as M consisting of:

- the replacement of s by t for $(s, t) \in M$,
- the deletion of s from S for $s \in V(S) \setminus M^{(1)}$,
- the insertion of t into S for $t \in V(T) \setminus M^{(2)}$.

2. Let an edit script $E = \langle e_1, \dots, e_n \rangle \in \mathcal{E}(S, T)$. Then, there exists a series of trees $\langle T_0, \dots, T_n \rangle$ such that $T_0 = S, T_n = T$, and the i -th edit operation $e_i = (s_i \mapsto t_i)$ transforms T_{i-1} into T_i for $i \in \{1, \dots, n\}$. We prove the lemma by induction on n .

If $n = 0$, the edit script $E = \langle \rangle$ corresponds to the Tai mapping $M = \{(s, s) \mid s \in S\}$, i.e. an isomorphic mapping from S to S . Thus, we have $\text{cost}(E) = \text{cost}(M) = 0$.

If $n = 1$, the cost of edit script $E = \langle (t \mapsto t') \rangle$ exactly corresponds to a Tai mapping M . Therefore, $\text{cost}(E) = \text{cost}(M) = d(t, t')$.

If $n \geq 2$, by the induction hypothesis there exists a Tai mapping M_1 from T_0 to T_{n-1} such that $\text{cost}(M_1) \leq \text{cost}(\langle e_1, \dots, e_{n-1} \rangle)$. Consider the transformation $T_{n-1} \xrightarrow{e_n} T_n$ via an edit operation $e_n = (t \mapsto t')$, where either t or t' (not both) can be a null node ε . Then, there exists a Tai mapping $M_2 = M' \cup \{(t, t')\}$ from T_{n-1} to T_n such that M' is an isomorphic mapping from T_{n-1} to T_n except for a difference by the effect of $\{(t, t')\}$. Hence, $\text{cost}(M_2) = d(t, t')$. Now we have a Tai mapping $M = M_2 \circ M_1$ from S to T . By Lemma 2.40, the following holds.

$$\text{cost}(M) \leq \text{cost}(M_1) + \text{cost}(M_2) \leq \text{cost}(\langle e_1, \dots, e_{n-1} \rangle) + d(t, t') = \text{cost}(E).$$

It follows by induction that the assertion holds for all edit scripts E . ■

By using Lemma 2.49, Tai showed that the tree edit distance in Definition 2.32 is reduced into the cost minimization problem of tree mappings.

Theorem 2.50 (Theorem 3.1 in [Tai79]) For two trees S and T , the following holds.

$$\mathbf{D}^{\text{Tai}}(S, T) = \min_{E \in \mathcal{E}(S, T)} \text{cost}(E) = \min_{M \in \mathcal{M}^{\text{Tai}}(S, T)} \text{cost}(M).$$

Proof. Straightforward from Lemma 2.49. ■

This theorem bridges the gap between the operational and declarative definitions of Tai distance. If the cost of a Tai mapping M is minimum, this mapping M is referred to as an *optimal Tai mapping*. Note that Tai mapping is a natural representation of root-editable operations in Remark 2.2.

By using the transitivity of Tai mappings, Tai showed that Tai distance is a metric.

Corollary 2.51 (Metricity of Tai Distance) Tai distance is a metric.

Proof. Since there exists an isomorphic mapping between the same two trees, and Tai mapping is symmetric, then it is obvious that for any trees S and T ,

$$\mathbf{D}^{\text{Tai}}(T, T) = 0 \quad \text{and} \quad \mathbf{D}^{\text{Tai}}(S, T) = \mathbf{D}^{\text{Tai}}(T, S).$$

Since tree mapping is subadditive by Lemma 2.40, and Tai mapping is transitive by Proposition 2.48, it follows from Proposition 2.45 and Theorem 2.50 that Tai distance is transitive, i.e. for any trees R, S , and T , we have

$$\mathbf{D}^{\text{Tai}}(R, T) \leq \mathbf{D}^{\text{Tai}}(R, S) + \mathbf{D}^{\text{Tai}}(S, T).$$

Therefore, Tai distance is a metric. ■

2.5.3 Approximate Common Subforest Problem

There is an alternative view in defining Tai distance as in the string case in Section 2.2.3. In Definition 2.32, for given two trees, we apply all edit operations to the first tree to obtain the second. In the alternative view, we use only *replacements* and *deletions* as the elementary edit operations, and define the Tai distance between two trees as the minimum cost of edit operations to transform two trees into a common third forest, i.e. we redefine $\mathcal{E}(S, T)$ as the all possible edit operations to obtain a third common forest by applying replacements and deletions to two trees S and T . In other words, this problem is to find an *approximate common subforest* shared by two trees with the minimum cost of edit operations without *insertions*. In this case, we permit the edit operation with edit signature $\varepsilon \mapsto \varepsilon$. This operation changes nothing and is called an *identity edit operation*.

The problem of computing the edit distance between two trees along with an approximate common subforest of the minimum cost of edit operations is called the *approximate common subforest problem*. It is easy to show that the approximate common subforest problem is equivalent to the tree edit problem in the computation of tree edit distance since any deletion of a node x from the second tree has its complementary operation, the insertion of x into the first tree with the edit signature $\varepsilon \mapsto x$.

From the viewpoint of Tai mapping, this problem is clear. For a Tai mapping M between two trees S and T , approximate common subforests are both $S[M^{(1)}]$ and $T[M^{(2)}]$. From the definition of Tai mapping, these two forests are isomorphic if the effect of replacements are ignored. If replacements are not used, $S[M^{(1)}]$ and $T[M^{(2)}]$ are isomorphic.

Example 2.52 **Figure 2.12** shows a Tai mapping between S and T , and a common subforest pattern F of S and T . ■

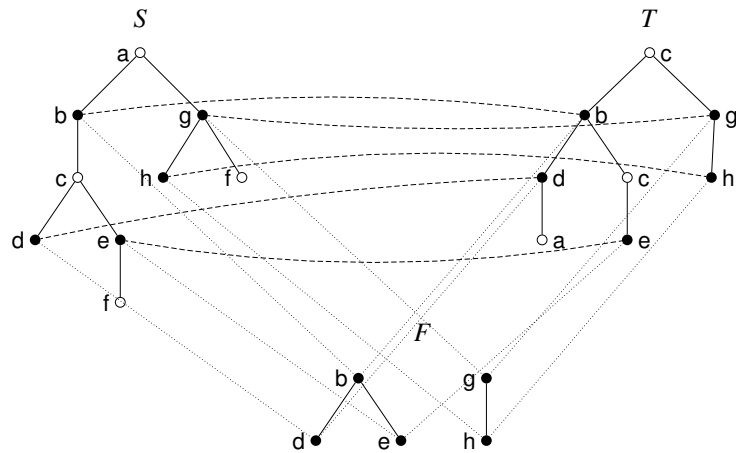


Figure 2.12. *Approximate Common Subforest*

Table 2.5. *Some classes of approximate tree matching*

Reference	Tree mapping / Distance	Class \mathcal{C}	Section
[Tai79]	Tai	TAI	§2.4
[JWZ95]	alignable / alignment	ALN	§2.7
[TT88]	structure-preserving	SP	§2.8.1
[OT88, Tan84]	strongly structure-preserving	SP ^b	§2.8.2
[Lu79]	Lu	LU	§2.8.3
[Zha95]	constrained	CST	§2.8.4
[Ric97]	structure-respecting	SR	§2.8.5
[LST01]	less-constrained	CST [#]	§2.8.6
[Sel77]	top-down (LCST)	TOP	§2.9.1
[Val01]	bottom-up	BOT	§2.9.2

2.6 Variants of The Tree Edit Problem

A variety of tree edit distance measures have been proposed other than Tai distance. Most of them are defined by imposing a certain restriction on the edit operations or the tree mappings of Tai distance.

There are two major motivations for restricting Tai mapping. The first is to improve the computation cost of the edit problem. The second is to tailor a tree mapping for specific applications, since Tai mapping may be too general for certain applications such as comparing parse trees, taxonomies, and more structure sensitive distance measures are required in these applications. In the following sections, we give a cursory review on some of important classes of tree mappings in tree edit distance, and related problems. Once a class definition of tree mappings is given, we can define the distance measure by using the condition based on Definition 2.43. Then, we use mainly a declarative definition if it is known.

Table 2.5 supplies terms for classes of approximate tree matching based on the optimization of edit scripts or tree mappings. Some of the terms are slightly modified from the originals due to uniformity.

Before reviewing a variety classes of approximate tree matching, we define two representative cost functions commonly used in various distance measures.

2.6.1 Unit Costs

If the tree edit distance between two trees is defined as *the number of edit operations* to transform one tree into another, the cost function d is given by

$$d(x, y) = \begin{cases} 0 & \text{if } l(x) = l(y), \\ 1 & \text{if } l(x) \neq l(y). \end{cases}$$

Table 2.6. Computational complexity of alignment problem

Trees	Reference	Degree unbounded		Degree bounded	
		Time	Space	Time	Space
ordered	[JWZ95]	$O(n_1 n_2 (d_1 + d_2)^2)$	$O(n_1 n_2 (d_1 + d_2))$	$O(n^2)$	$O(n^2)$
	[WZ03]	$O(n_1^2 n_2 (d_1 + d_2)^2)$	$O((\log n_1) n_2 (d_1 + d_2) d_1)$	$O(n^3)$	$O(n \log n)$
	[WZ05]	$O(n_1 n_2 (d_1 + d_2)^2)$	$O((\log n_1) n_2 (d_1 + d_2) d_1)$	$O(n^2)$	$O(n \log n)$
	[Jan03]	$O(k^2 n (\log n + d^3))$		$O(k^2 n \log n)$	
unordered	[JWZ95]	MAX SNP-hard		polynomial [†]	
	[FA06] (unit costs, degree and alphabet size bounded)			$O(\gamma^K n)$	

n_i : size of tree T_i for $i \in \{1, 2\}$, and assume that $n_1 \leq n_2$ and $n = n_2$, $d_i = \deg(T_i)$ for $i \in \{1, 2\}$.

k : fixed upper bound of the number of inserted gap symbols.

K : fixed upper bound of the unit-cost alignment distance.

γ : parameter determined by d_i (e.g. $\gamma \leq 4.45$ for unordered binary trees).

[†]For unordered binary trees, $O(n^2)$ time and space.

We refer to the \mathcal{C} -distance based on the cost function d as \mathcal{C} -distance with *unit costs* or simply *unit-cost* \mathcal{C} -distance, and we denote by $D_1^{\mathcal{C}}(S, T)$ the unit-cost \mathcal{C} -distance between S and T . The cost of a tree mapping M between two trees S and T is simplified as follows:

$$\begin{aligned} \text{cost}(M) &= |\{(x, y) \in M \mid x \neq y\}| + |V(S) \setminus M^{(1)}| + |V(T) \setminus M^{(2)}| \\ &= |\{(x, y) \in M \mid x \neq y\}| + |S| + |T| - 2 \cdot |M|. \end{aligned}$$

2.6.2 Largest Common Subforest Patterns

On the other hand, if the tree edit distance between two trees is defined as *the number of edit operations* to transform one tree into another *without replacements*, the cost function d is given by

$$d(x, y) = \begin{cases} 0 & \text{if } l(x) = l(y), \\ 1 & \text{if } l(x) \neq l(y) \text{ and } (l(x) = '-' \text{ or } l(y) = '-'), \\ \infty & \text{otherwise (label replacement).} \end{cases}$$

As in the case of strings (Section 2.2.3), if we need just the distance value, it is enough to let the cost of replacement be more than or equal to 2. By computing the \mathcal{C} -distance between two trees S and T based on the cost function d , we can find a *largest common subforest pattern* of S and T . (Tai refers to it as a *largest common substructure* between S and T [Tai79].) We refer to the distance as \mathcal{C} -distance with *LCS costs* or simply *LCS-cost* \mathcal{C} -distance, and we denote by $D_{\text{lcs}}^{\mathcal{C}}(S, T)$ the LCS-cost \mathcal{C} -distance between S and T .

2.7 Alignment of Trees and Alignment Distance

The *alignment of trees* was introduced by Jiang *et al.* [JWZ95], as a natural extension of the alignment of strings, in search of better comparison methods for RNA secondary structures. While the Tai edit problem can be regarded as the problem of finding an approximate common subforest of two trees, the alignment of trees can be regarded as the problem of finding an *approximate common supertree* of two trees.

We summarize the proposed algorithms and computational complexities for computing alignment of trees in **Table 2.6**.

2.7.1 Operational Definitions

The definition of alignment of trees was given in an operational way [JWZ95] as follows.

Definition 2.53 (Alignment of Trees [JWZ95]) An *alignment* of two trees S and T is obtained by the following two steps:

1. Insert new nodes with gap symbol “-” into S and T so that the following two conditions are satisfied:
 - (i) two resulting trees S' and T' are isomorphic if labels are ignored, and let ϕ be an isomorphic mapping from S' to T'
 - (ii) for any node $s \in S'$ labeled with a gap symbol, the node $\phi(s) \in T'$ is not labeled with a gap symbol.
2. Collect the pairs of nodes as follows:

$$A = \{(s, \phi(s)) \in V(S') \times V(T') \mid s \in S'\}.$$

We refer to A as an alignment of S and T . The tree obtained by relabeling all nodes in S' with $(l(s), l(\phi(s)))$ for $s \in S'$ is called an *aligned tree*.


The cost of an alignment A is defined as the sum of the costs of all pairs of aligned nodes :

$$\text{cost}(A) = \sum_{(s,t) \in A} d(s, t) = \sum_{(s,t) \in A} d_l(l(s), l(t)).$$

An *optimal alignment* is an alignment that minimizes the cost over all possible alignments. An *alignment distance* is the cost of an optimal alignment. We denote the set of all possible alignments between two trees S and T by $\mathcal{A}(S, T)$. Then, the *alignment distance* between two trees S and T is given as follows.

$$\mathbf{D}^{\text{ALN}}(S, T) = \min_{A \in \mathcal{A}(S, T)} \text{cost}(A).$$

The problem of computing an optimal alignment or alignment distance is referred to as the *alignment problem* in trees.

 **Tree Mapping for Alignment of Trees.** In Definition 2.53, alignment distance is not defined as the minimum cost of tree mappings as in the case of Tai distance. We refer to the class of tree mappings for determining alignment distance as *alignable mappings*. For any alignment $A \in \mathcal{A}(S, T)$, an alignable mapping M is given by

$$M = \{(s, t) \in V(S) \times V(T) \mid (s, t) \in A\}.$$

The explicit condition of alignable mappings had not been identified for ten years since Jiang *et al.* first introduced the notion of alignment of trees in an operational way [JWZ95]. In Section 4.6, we reveal the condition of alignable mappings.

Example 2.54 Consider the unit-cost alignment distance between two ordered trees S and T in **Figure 2.13**. Then, we have $\mathbf{D}_1^{\text{ALN}}(S, T) = 4$ since the minimum cost is computed by

$$d_l(\mathbf{a}, \mathbf{a}) + d_l(\mathbf{b}, -) + d_l(\mathbf{c}, \mathbf{c}) + d_l(-, \mathbf{f}) + d_l(\mathbf{d}, \mathbf{d}) + d_l(-, \mathbf{e}) + d_l(\mathbf{e}, -) = 4.$$

Figure 2.14(left) shows the alignable mapping corresponding to the alignment in **Figure 2.13**, and by overlaying the node pairs in the mapping we obtain the same aligned tree **Figure 2.14**(right) as in **Figure 2.13**.

An alignment of two trees is not uniquely determined. **Figure 2.15** shows another alignable mapping between S and T , and its aligned tree.

Figure 2.16(left) shows a Tai mapping between S and T . This mapping is not an alignable mapping since by overlaying the node pairs in the mapping we obtain an acyclic directed graph in **Figure 2.16**(right), and it is not a tree. It is obvious from this observation that alignable mappings are in a proper subclass of Tai mappings. ■

Alignment Problem as a Restricted Tree Edit Problem

The alignment problem is viewed as a restricted tree edit problem in which all the insertions precede all the deletions. In the alignment problem for strings, the set of edit operations gives the same distance regardless of the order of operations, while the order may cause a big difference in the tree edit problem.

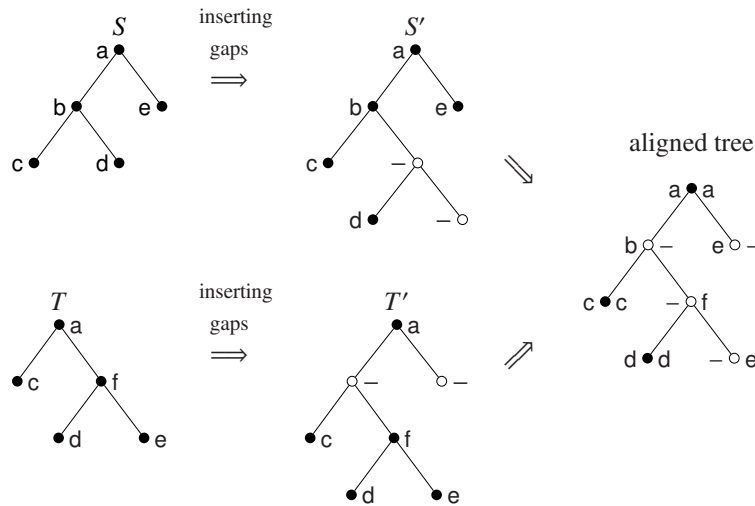


Figure 2.13. Alignment of trees between S and T

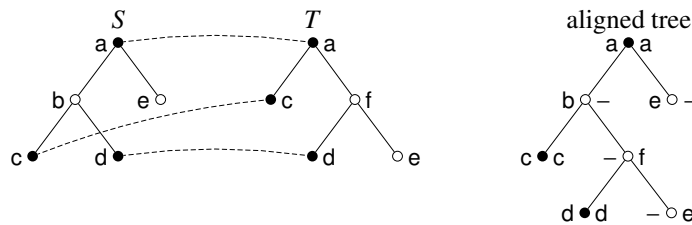


Figure 2.14. Alignable mapping and its aligned tree

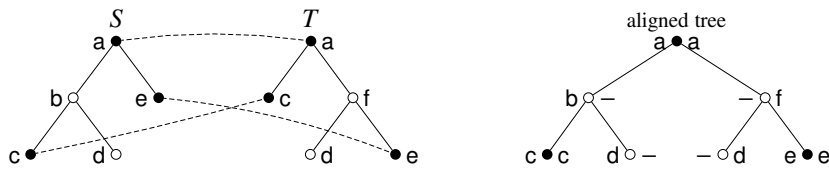


Figure 2.15. Another alignable mapping and its aligned tree

Example 2.55 For two trees S and T in **Figure 2.17**, we assume unit costs. Then, the alignment problem is solved by applying all insertions before deletions, and an optimal edit script is

$$E = \langle \varepsilon \mapsto f, \varepsilon \mapsto e, e \mapsto \varepsilon, b \mapsto \varepsilon \rangle.$$

There is no shorter path between S and T under the restriction. On the other hand, if there is no such restriction, then we have a shorter path with an optimal edit script

$$E = \langle b \mapsto \varepsilon, \varepsilon \mapsto f \rangle$$

as shown in **Figure 2.18**. ■

2.7.2 Approximate Common Supertree Problem

There is yet another alternative view of the alignment problem. First, let us define the notion of supertree. For a tree T , a *supertree* of T is a tree obtained by inserting an arbitrary number of nodes into T .

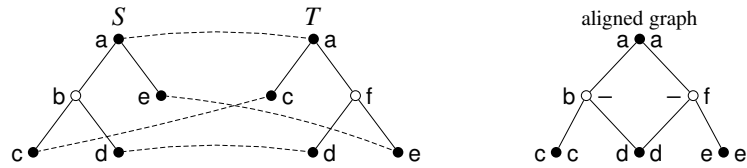


Figure 2.16. Non-alignable mapping

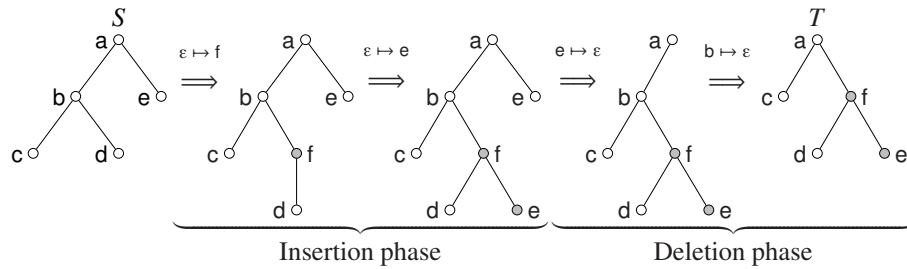


Figure 2.17. Alignment problem as an edit problem

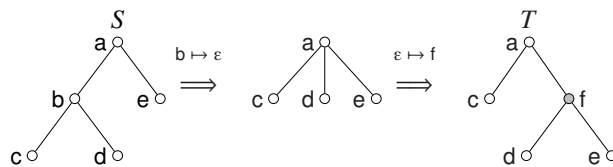


Figure 2.18. Tai edit problem

In the alternative view, we use only *replacements* and *insertions* as the elementary edit operations, and define the alignment distance as the minimum cost of edit operations to transform two trees into a common third tree. In other words, this problem is to find an approximate common supertree shared by two trees with the minimum cost of edit operations without *deletions*. We permit the identity edit operation as in the approximate common subtree problem. For example, in Figure 2.15, the tree to the right is obtained from two tree S and T by inserting two nodes respectively.

The problem of computing the alignment distance between two trees along with an approximate common supertree of the minimum cost of edit operations is called the *approximate common supertree problem*.

For strings, the approximate common *supersequence* problem is equivalent to the approximate common *subsequence* problem. For trees, both are, however, different as already seen.

2.7.3 Algorithms for Ordered Trees

For two trees T_1 and T_2 , we set $n = \max\{|T_1|, |T_2|\}$ and $d = \max\{\deg(T_1), \deg(T_2)\}$. Jiang *et al.* first proposed an $O(n^2 d^2)$ -time and $O(n^2 d^2)$ -space algorithm [JWZ95] for computing alignment distance for ordered trees. Later, Wang and Zhao [WZ03] improved the space complexity by sacrificing the time complexity for the case of $d \ll n$ as often seen in RNA secondary structures. The algorithm runs in $O(n^3 d^2)$ time and $O(n \log n \cdot d^2)$ space. Recently, Wang and Zhao achieved a further improvement by proposing a new algorithm which runs in $O(n^2 d^2)$ time and $O(n \log n \cdot d^2)$ space.

Jansson and Lingas proposed a *fixed-parameter algorithm* [JL03, Jan03] for ordered trees. The algorithm runs in $O(k^2 n (\log n + d^3))$ time for a fixed upper bound of the number of inserted nodes labeled with gap symbols k , i.e. this algorithm returns the alignment distance between given two trees if there exists an optimal alignment with at most k nodes labeled with gap symbols.

$$\begin{aligned}
D(\emptyset, \emptyset) &= 0 \\
D(F, \emptyset) &= \sum_{T \in F} D_T(T, \emptyset), & D(\emptyset, F) &= \sum_{T \in F} D_T(\emptyset, T) \\
D_T(v(F), \emptyset) &= D(F, \emptyset) + d(v, \varepsilon), & D_T(\emptyset, v(F)) &= D(\emptyset, F) + d(\varepsilon, v) \\
D_T(v_1(F_1), v_2(F_2)) &= \\
\min \begin{cases} D_T(\emptyset, v_2(F_2)) + \min_{T \in F_2} \{D_T(v_1(F_1), T) - D_T(\emptyset, T)\}, & \langle \varepsilon \mapsto v_2 \rangle \\ D_T(v_1(F_1), \emptyset) + \min_{T \in F_1} \{D_T(T, v_2(F_2)) - D_T(T, \emptyset)\}, & \langle v_1 \mapsto \varepsilon \rangle \\ D(F_1, F_2) + d(v_1, v_2), & \langle v_1 \mapsto v_2 \rangle \end{cases} & (2.4a) \\
D(F_1 \bullet T_1, F_2 \bullet T_2) &= \min \begin{cases} D(F_1, F_2 \bullet T_2) + D_T(T_1, \emptyset) \\ D(F_1 \bullet T_1, F_2) + D_T(\emptyset, T_2) \\ D(F_1, F_2) + D_T(T_1, T_2) \\ D'(F_1 \bullet T_1, F_2 \bullet T_2) \end{cases} & (2.4b) \\
D'(F_1 \bullet v_1(F'_1), F_2 \bullet v_2(F'_2)) &= D'(T_1^1 \bullet \dots \bullet T_1^m, T_2^1 \bullet \dots \bullet T_2^n) = \\
\min \begin{cases} \min_{1 \leq i \leq m} \{D(T_1^1 \bullet \dots \bullet T_1^{i-1}, F_2) + D(T_1^i \bullet \dots \bullet T_1^m, F'_2)\} + d(\varepsilon, v_2) \\ \min_{1 \leq i \leq n} \{D(F_1, T_2^1 \bullet \dots \bullet T_2^{i-1}) + D(F'_1, T_2^i \bullet \dots \bullet T_2^n)\} + d(v_1, \varepsilon) \end{cases} & (2.4c)
\end{aligned}$$

Figure 2.19. Recurrences for computing alignment distance for ordered trees

Jiang-Wang-Zhang's Algorithm

The algorithm for computing alignment distance [JWZ95] for ordered trees was proposed based on the recurrences in **Figure 2.19**, where $D(F_1, F_2)$ denotes the alignment distance between two forests F_1 and F_2 , and $D_T(T_1, T_2)$ denotes the alignment distance between two trees T_1 and T_2 . **Figure 2.20** and **Figure 2.21** illustrate the intuitive meanings of Eq.(2.4a) and Eq.(2.4c) respectively. In Algorithm 2.4, we show the algorithm for computing alignment distance.

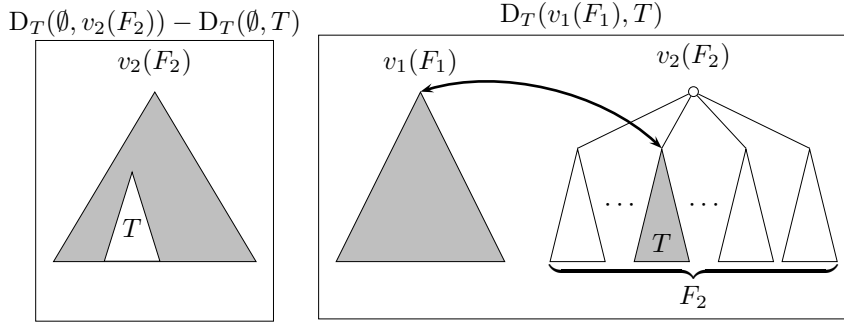


Figure 2.20. $D_T(\emptyset, v_2(F_2)) + \min_{T \in F_2} \{D_T(v_1(F_1), T) - D_T(\emptyset, T)\}$ in Eq.(2.4a)

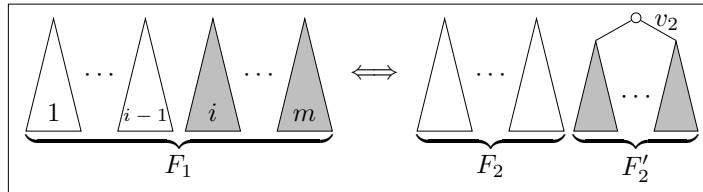


Figure 2.21. $D(T_1^1 \bullet \dots \bullet T_1^{i-1}, F_2) + D(T_1^i \bullet \dots \bullet T_1^m, F'_2)$ in Eq.(2.4c)

An optimal alignable mapping (optimal alignment) can be basically computed by tracing back the resulting arrays D and D_T obtained by TREEALIGN and FORESTALIGN.

Algorithm 2.4 Alignment distance for ordered trees

```

procedure TREEALIGN( $T_1, T_2$ )
   $D(\emptyset, \emptyset) \leftarrow 0$ 
  foreach  $v \in T_1$  in postorder do
    let  $F_1$  be a forest such that  $T_1(v) = v(F_1)$ 
     $D(F_1, \emptyset) \leftarrow \sum_{T \in F_1} D_T(T, \emptyset)$ 
     $D_T(v(F_1), \emptyset) \leftarrow D(F_1, \emptyset) + d(v, \varepsilon)$ 
  foreach  $v \in V(T_2)$  in postorder do
    let  $F_2$  be a forest such that  $T_2(v) = v(F_2)$ 
     $D(\emptyset, F_2) \leftarrow \sum_{T \in F_2} D_T(\emptyset, T)$ 
     $D_T(\emptyset, v(F_2)) \leftarrow D(\emptyset, F_2) + d(\varepsilon, v)$ 
  for  $v_1 \in T_1$  in postorder do
    let  $F_1 = T_1^1 \bullet \dots \bullet T_1^m$  be a forest such that  $T_1(v_1) = v_1(F_1)$ 
    for  $v_2 \in T_2$  in postorder do
      let  $F_2 = T_2^1 \bullet \dots \bullet T_2^n$  be a forest such that  $T_2(v_2) = v_2(F_2)$ 
      for  $i \leftarrow 1$  to  $m$  do
        FORESTALIGN( $T_1^i \bullet \dots \bullet T_1^m, F_2$ )
      for  $j \leftarrow 1$  to  $n$  do
        FORESTALIGN( $F_1, T_2^j \bullet \dots \bullet T_2^n$ )
       $D_T(v_1(F_1), v_2(F_2)) = \min \begin{cases} D_T(\emptyset, v_2(F_2)) + \min_{T \in F_2} \{D_T(v_1(F_1), T) - D_T(\emptyset, T)\} \\ D_T(v_1(F_1), \emptyset) + \min_{T \in F_1} \{D_T(T, v_2(F_2)) - D_T(T, \emptyset)\} \\ D(F_1, F_2) + d(v_1, v_2) \end{cases}$ 
    return  $D_T(T_1, T_2)$ 
end

procedure FORESTALIGN( $T_1^1 \bullet \dots \bullet T_1^m, T_2^1 \bullet \dots \bullet T_2^n$ )
  for  $i \leftarrow 1$  to  $m$  do
     $D(T_1^1 \bullet \dots \bullet T_1^i, \emptyset) \leftarrow D(T_1^1 \bullet \dots \bullet T_1^{i-1}, \emptyset) + D_T(T_1^i, \emptyset)$ 
  for  $i \leftarrow 1$  to  $n$  do
     $D(\emptyset, T_2^1 \bullet \dots \bullet T_2^i) \leftarrow D(\emptyset, T_2^1 \bullet \dots \bullet T_2^{i-1}) + D_T(\emptyset, T_2^i)$ 
  for  $i \leftarrow 1$  to  $m$  do
    for  $j \leftarrow 1$  to  $n$  do
      let  $F_k \bullet v_k(F'_k) = T_k^1 \bullet \dots \bullet T_k^i$  for  $k \in \{1, 2\}$ 
       $D(T_1^1 \bullet \dots \bullet T_1^i, T_2^1 \bullet \dots \bullet T_2^j) =$ 
       $\min \begin{cases} D(T_1^1 \bullet \dots \bullet T_1^{i-1}, T_2^1 \bullet \dots \bullet T_2^j) + D_T(T_1^i, \emptyset) \\ D(T_1^1 \bullet \dots \bullet T_1^i, T_2^1 \bullet \dots \bullet T_2^{j-1}) + D_T(\emptyset, T_2^j) \\ D(T_1^1 \bullet \dots \bullet T_1^{i-1}, T_2^1 \bullet \dots \bullet T_2^{j-1}) + D_T(T_1^i, T_2^j) \\ \min_{1 \leq k \leq i} \{D(T_1^1 \bullet \dots \bullet T_1^{k-1}, F_2) + D(T_1^k \bullet \dots \bullet T_1^i, F'_2)\} + d(\varepsilon, v_2) \\ \min_{1 \leq k \leq j} \{D(F_1, T_2^1 \bullet \dots \bullet T_2^{k-1}) + D(F'_1, T_2^k \bullet \dots \bullet T_2^j)\} + d(v_1, \varepsilon) \end{cases}$ 
    end

```

We, however, need to take notice that Jiang-Wang-Zhang's algorithm is designed for computing just one optimal alignable mapping even if there exists more than one. Although Zhang-Shasha's algorithm for Tai distance also computes just one optimal Tai mapping, the resulting arrays implicitly contain all possible optimal Tai mappings. Hence, we can enumerate all the possible optimal Tai mappings from the resulting arrays by traceback.

On the other hand, the resulting arrays by Jiang-Wang-Zhang's algorithm excludes some possible alignable mappings in the first place since these excluded mappings are always replaceable with alternative mappings with the same costs from the viewpoint of cost optimization. Therefore, in a precise sense, the recurrences used in Jiang-Wang-Zhang's algorithm do not mathematically correspond to the definition of alignment of trees, and it computes alignment distance in a subclass of alignable mappings. This is a slight difference in recurrences or implementation, but may cause a difference in the definition of tree mappings (See Section 4.8.1).

We do not enter into the detail of the correctness of all the recurrences in Figure 2.19. However, we sketch the proof of the correctness of Eq.(2.4a) since the exclusiveness of a tree mapping arises from Eq.(2.4a).

Proof of the correctness of Eq.(2.4a) [JWZ95, Lemma 2]. Consider an aligned tree of an optimal alignment A of $T_1 = v_1(F_1)$ and $T_2 = v_2(F_2)$. We assume that

$$\begin{aligned}\Delta_1 &= \{x \notin T_1 \mid (x, y) \in A \text{ for some } y \in T_2\} \\ \Delta_2 &= \{y \notin T_2 \mid (x, y) \in A \text{ for some } x \in T_1\}\end{aligned}$$

It suffices to consider the following four cases.

1. $(v_1, v_2) \in A$: The nodes v_1 and v_2 are aligned. Hence, the rest of $v_1(F_1)$ and $v_2(F_2)$, i.e. two forests F_1 and F_2 , need to be aligned, i.e.

$$D_T(v_1(F_1), v_2(F_2)) = D(F_1, F_2) + d(v_1, v_2).$$

2. $(v_1, w_2) \in A$ for some $w_2 \in \Delta_2$, and $(w_1, v_2) \in A$ for some $w_1 \in \Delta_1$: Recall that w_1 and w_2 are labeled with the same gap symbol. Hence, due to the metricity of d , the cost cannot be better than the case 1, i.e. $d_l(l(v_1), l(v_2)) \leq d_l(l(v_1), -) + d_l(-, l(v_2))$. Hence, we may ignore this case for computing the minimum cost.

3. $(v_1, w_2) \in A$ for some $w_2 \in \Delta_2$, and $(w_1, v_2) \notin A$ for any $w_1 \in \Delta_1$: The root of the aligned tree is obtained by aligning v_1 and w_2 , and the node v_2 must be aligned with a node in $T \in F_1$. Hence,

$$D_T(v_1(F_1), v_2(F_2)) = D_T(v_1(F_1), \emptyset) + \min_{T \in F_1} \{D_T(T, v_2(F_2)) - D_T(T, \emptyset)\}.$$

4. $(w_1, v_2) \in A$ for some $w_1 \in \Delta_1$, and $(v_1, w_2) \notin A$ for any $w_2 \in \Delta_2$: Symmetric to the case 3. ■

From this proof, it is clear that the case 3 is ignored in the recurrences in Figure 2.13. If this case is completely excluded, in a precise sense, the recurrences in Figure 2.13 end up with mathematically mismatching the definition of alignment of trees. In addition, this proof assumes the metricity of the cost function d (we see in Corollary 2.59 that alignment distance is not a metric even if d is a metric), while Zhang-Shasha's algorithm does not assume it. By adding the case 3 to Eq.(2.4a), we have the following recurrence mathematically equivalent to the definition of alignable mappings.

$$D_T(v_1(F_1), v_2(F_2)) = \min \begin{cases} D_T(\emptyset, v_2(F_2)) + \min_{T \in F_2} \{D_T(v_1(F_1), T) - D_T(\emptyset, T)\} \\ D_T(v_1(F_1), \emptyset) + \min_{T \in F_1} \{D_T(T, v_2(F_2)) - D_T(T, \emptyset)\} \\ D(F_1, F_2) + d(v_1, v_2) \\ \boxed{D(F_1, F_2) + d(v_1, \varepsilon) + d(\varepsilon, v_2)} \end{cases} \quad (2.5)$$

For enumerating all the alignments between two trees, we need to use this recurrence, or implement some supplementary procedure in traceback.

Example 2.56 We assume the following cost function:

$$d(x, y) = \begin{cases} 0 & \text{if } l(x) = l(y), \\ 1 & \text{if } l(x) \neq l(y) \text{ and } (l(x) = "-" \text{ or } l(y) = "-"), \\ 3 & \text{otherwise (label replacement).} \end{cases}$$

There are two possibilities of optimal alignable mappings as shown in Figure 2.19(a) and (b). Jiang-Wang-Zhang's algorithm, however, does not consider the case of (b). ■

Complexity. We here estimate the time complexity of Jiang-Wang-Zhang's algorithm. Let T_1 and T_2 be two given trees. For each pair of nodes $v_1 \in T_1$ and $v_2 \in T_2$, the procedure FORESTALIGN runs in

$$O(|\text{ch}(v_1)| \cdot |\text{ch}(v_2)| \cdot (|\text{ch}(v_1)| + |\text{ch}(v_2)|)) \text{ time,}$$

and then the procedure TREEALIGN runs in

$$O(|\text{ch}(v_1)| \cdot |\text{ch}(v_2)| \cdot (|\text{ch}(v_1)| + |\text{ch}(v_2)|)^2) \text{ time.}$$

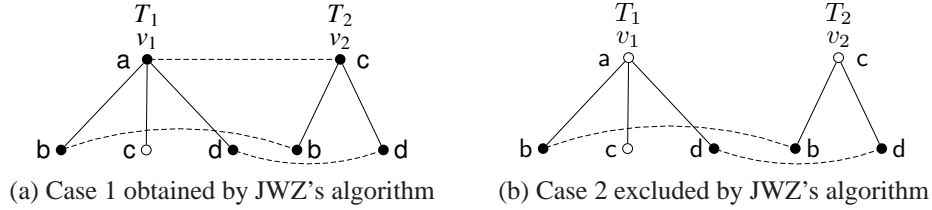


Figure 2.22. Two optimal alignable mappings

Therefore, the time complexity of Algorithm 2.4 is

$$\begin{aligned}
 & \sum_{v_1 \in T_1} \sum_{v_2 \in T_2} O\left(|\text{ch}(v_1)| \cdot |\text{ch}(v_2)| \cdot (|\text{ch}(v_1)| + |\text{ch}(v_2)|)^2\right) \\
 & \leq \sum_{v_1 \in T_1} \sum_{v_2 \in T_2} O\left(|\text{ch}(v_1)| \cdot |\text{ch}(v_2)| \cdot (\deg(T_1) + \deg(T_2))^2\right) \\
 & \leq O\left(\left(\sum_{v_1 \in T_1} |\text{ch}(v_1)|\right) \cdot \left(\sum_{v_2 \in T_2} |\text{ch}(v_2)|\right) \cdot (\deg(T_1) + \deg(T_2))^2\right) \\
 & \leq O\left(|T_1| \cdot |T_2| \cdot (\deg(T_1) + \deg(T_2))^2\right).
 \end{aligned}$$

Amalgamation of trees. It seems that alignment of trees can be applied to amalgamating two trees into one supertree based on the similarity. However, we have yet to solve the problem of determining one aligned tree out of more than one candidate. It is obvious that the way of amalgamation from an obtained alignment is not unique even for strings. The amalgamation of trees has, however, a more complex ambiguity than strings as shown in the following example.

Example 2.57 Consider two trees S and T , and an alignment A depicted by dashed lines in **Figure 2.23**. By overlaying pairs of nodes in A with preserving any order in S and T , we have two possible amalgamations R_1 and R_2 . ■

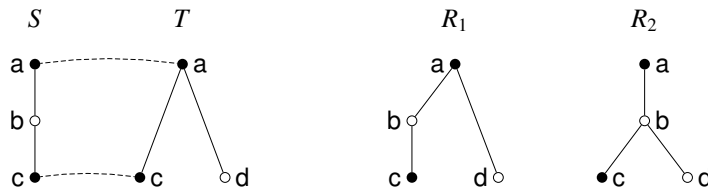


Figure 2.23. Two aligned trees obtained by using the same alignment

2.7.4 Algorithms for Unordered Trees

Jiang *et al.* proved that the alignment problem for unordered trees is MAX SNP-hard if either of given two trees has arbitrary degrees [JWZ95]. Jiang *et al.* [JWZ95] also presented a polynomial-time algorithm for unordered trees with bounded degree (the degree of a node is defined as the number of its children) by giving the same structure of recurrences in Figure 2.19. In particular, for unordered binary trees, an $O(n_1 n_2)$ -time and -space algorithm was shown in [JWZ95], where n_i for $i \in \{1, 2\}$ is the size of tree T_i .

Fukagawa and Akutsu proposed a *fixed-parameter algorithm* [FA06] for unit-cost alignment of unordered trees with bounded degree and a bounded size alphabet. This algorithm runs in $O(\gamma^k n)$ time, where $n = \max\{|T_1|, |T_2|\}$ for given two trees T_1 and T_2 , γ is a constant determined by $\max\{\deg(T_1), \deg(T_2)\}$, and k is a fixed upper bound of the number of inserted nodes labeled with gap symbols.

2.7.5 Alignable Mappings

Even without the explicit condition of alignable mappings, the following important property is known. By ALN we denote the class of alignable mapping. The following proposition leads to the fact that the alignment distance is *not* a metric.

Proposition 2.58 Alignable mapping is not transitive.

Proof. We show this proposition by a counterexample to transitivity. For trees R , S , and T , let $M_{RS} \in \mathcal{M}^{\text{ALN}}(R, S)$, $M_{ST} \in \mathcal{M}^{\text{ALN}}(S, T)$ be two alignable mappings. As shown in **Figure 2.24**, although M_{RS} and M_{ST} are alignable mappings, the composite $M_{ST} \circ M_{RS} \in \mathcal{M}(R, T)$ is not alignable. Since the composite of two alignable mappings is not necessarily alignable, alignable mapping is not transitive. ■

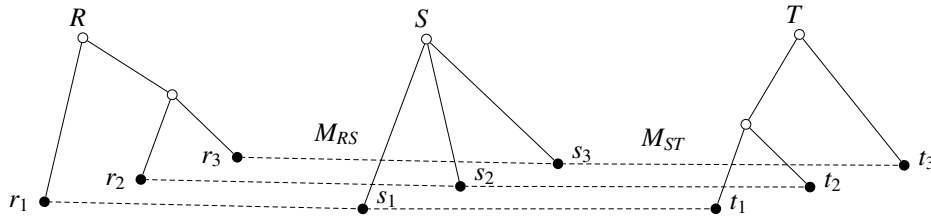


Figure 2.24. Counterexample to transitivity in alignable mappings

Although this intransitivity does not necessarily imply that alignable distance is not a metric, we immediately have the following corollary by using the same counterexample.

Corollary 2.59 Alignable distance is not a metric.

Proof. See the following counterexample. ■

Example 2.60 We assume unit costs. For trees R , S and T , consider three *optimal* alignable mappings $M_{RS} \in \mathcal{M}^{\text{ALN}}(R, S)$, $M_{ST} \in \mathcal{M}^{\text{ALN}}(S, T)$, and $M_{RT} \in \mathcal{M}^{\text{ALN}}(R, T)$ as shown in **Figure 2.25**. Then, we have

$$D_1^{\text{ALN}}(R, T) = 4, \quad D_1^{\text{ALN}}(R, S) = 1, \quad D_1^{\text{ALN}}(S, T) = 1.$$

Hence,

$$D_1^{\text{ALN}}(R, T) > D_1^{\text{ALN}}(R, S) + D_1^{\text{ALN}}(S, T).$$

It follows that alignable distance is not a metric.

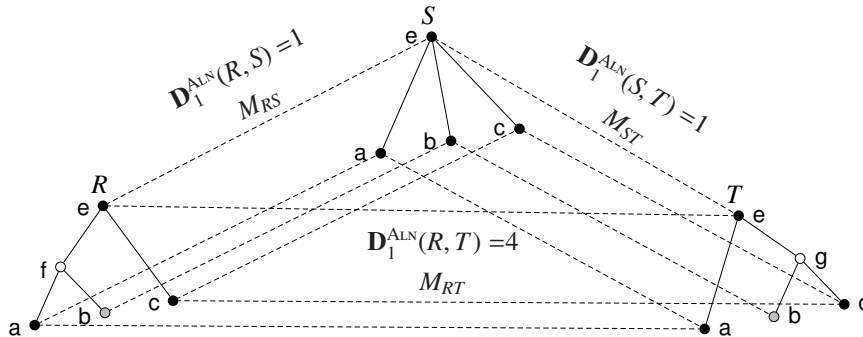


Figure 2.25. Non-metricity of alignable distance

For Tai distance, we have

$$D_1^{\text{Tai}}(R, T) = 2, \quad D_1^{\text{Tai}}(R, S) = 1, \quad D_1^{\text{Tai}}(S, T) = 1,$$

since Tai distance is a metric. ■

2.8 Structure Sensitive Distance

Tai distance and alignment distance are primarily defined in an operational way by using edit operations. In these two distance measures, the notion of tree mapping is rather used for characterizing the distance measures. In contrast, some distance measures between trees are primarily defined based on Definition 2.43 by giving subclass definitions of Tai mappings without explicit definitions of how to apply edit operations to trees. These classes consider not merely node-to-node mappings but also subtree-to-subtree mappings by tree mappings. In this section, we show these classes of tree mappings, and related distance measures. We summarize the complexities of these algorithms in **Table 2.7**.

Table 2.7. Computational complexity of structure sensitive distance problems

Trees		Class	Reference	Time	Space
ordered	SP	structure-preserving [†]	[TT88]	$O(n_1 n_2 l_2)$	$O(n_1 n_2)$
	SP ^b	strongly structure-preserving	[Tan84]	$O(n_1 n_2)$	$O(n_1 n_2)$
	LU	Lu	[Lu79]	$O(n_1 n_2)$	$O(n_1 n_2)$
	SR	structure-respecting	[Ric97]	$O(n_1 n_2 d_1 d_2)$ [‡]	$O(d_1 h_1 n_2)$ ^{††}
	CST	constrained	[Zha95]	$O(n_1 n_2)$	$O(n_1 n_2)$
	CST	constrained	[WZ05]	$O(n_1 n_2)$	$O((\log n_1) n_2)$ ^{‡‡}
	CST [#]	less-constrained	[LST01]	$O(n_1 n_2 d_1^3 d_2^3 d)$	
unordered	CST	constrained	[Zha96]	$O(n_1 n_2 d \log d)$	$O(n_1 n_2)$
	CST [#]	less-constrained	[LST01]	MAX SNP-hard ^{†††}	
	LU	Lu	[AYO ⁺ 03]	$O(n_1 n_2)$ ^{‡‡‡}	$O(n_1 n_2)$

For $i \in \{1, 2\}$, n_i : size of tree T_i , $h_i = \text{dep}(T_i)$, $\ell_i = |\text{leaves}(T_i)|$, $d_i = \text{deg}(T_i)$, $d = d_1 + d_2$

[†]SP-distance is asymmetric w.r.t. T_1 and T_2 .

[‡]The algorithm for SR-distance is exactly the same as the algorithm for CST-distance except for the complexity estimation.

^{††}For the computation of an optimal SR-mapping, $O(n_1 n_2)$ space is required.

^{‡‡}An optimal CST-mapping can also be computed with this space complexity.

^{†††}In fact, a more negative result is proven. There is no absolute polynomial-time approximation unless P=NP.

^{‡‡‡}This algorithm assumes degree-bounded trees.

2.8.1 Structure-Preserving Distance

Tanaka and Tanaka introduced a subclass of Tai mapping [TT82, Tan93, TT88, Tan95] by restricting the condition of Tai mapping. It was probably the first attempt at defining a distance measure between two trees by considering tree images mapped by a tree mapping.

Since the original definition of structure-preserving mapping is a slightly verbose, we modify the representation of the definition without changing the essential formalization.

The node of the rightmost leaf of a complete subtree of T rooted at $t \in T$ is denoted by $rl(t)$. Let M be a tree mapping from S to T . For $s \in S$, we define the *root image* of s under M as follows:

$$\mathbf{R}_M(s)^\dagger = \begin{cases} \text{lca}(\{M(x) \in T \mid x \leq_S s\}) & \text{if } \{x \in M^{(1)} \mid x \leq_S s\} \neq \emptyset, \\ \perp \text{ (undefined)} & \text{otherwise.} \end{cases}$$

Note that $\mathbf{R}_M(s)$ may not be well-defined for some $s \in S$. (If $\mathbf{R}_M(s) \neq \perp$, we say that $\mathbf{R}_M(s)$ is well-defined for s .) For a node $s \in S$, we refer to the complete subtree rooted at $\mathbf{R}_M(s)$ as the image of $S(s)$ under M . The structure-preserving mapping is represented as follows.

[†] $\mathbf{R}_M(s)$ is in effect the same as $r(s)$ in [TT82, TT88].

Definition 2.61 (Structure-Preserving Mapping [TT82, Tan93, TT88, Tan95])

For two ordered trees S and T , a Tai mapping M from S to T is *structure-preserving* if the following condition is satisfied:

$$\forall s_1, s_2 \in S \left[\mathbf{R}_M(s_1) \neq \perp \wedge \mathbf{R}_M(s_2) \neq \perp \implies [rl(s_1) \triangleleft_S s_2 \iff rl(\mathbf{R}_M(s_1)) \triangleleft_T \mathbf{R}_M(s_2)] \right].$$

This definition uses a property of left-to-right preorder such that $V(T(t)) = \{x \in T \mid t \triangleleft_T x \triangleleft_T rl(t)\}$ for any node $t \in T$. The definition implies that “two subtrees $S(s_1)$ and $S(s_2)$ are isolated (i.e. not overlapped) if and only if two subtrees $T(\mathbf{R}_M(s_1))$ and $T(\mathbf{R}_M(s_2))$ are isolated for any $s_1, s_2 \in S$.”

Remark 2.3 Note that the original definition of structure-preserving mapping was incomplete in [TT82] (in Japanese) and [TT88] (in English). It was, later, corrected by adding the condition $\forall (i_1, i_2), (j_1, j_2) \in M [i_1 \in \text{An}(i_2) \iff j_1 \in \text{An}(j_2)]$ to the definition in [TT82] or [TT88]. The notes [Tan93] and [Tan95] are complementary remarks of [TT82] (in Japanese), and [TT88] (in English) respectively for the correction.



Structure-preserving mapping & isolated-subtree or constrained tree mapping. The idea of structure-preserving mapping has been widely prevailing due to its importance since it was proposed. But, the slightly complicated formulation of this idea has been occasionally led to a misunderstanding of the structure-preserving mapping. For example, [WZ01] and [Val01] regard the structure-preserving mapping and the constrained mapping [Zha95] as the same class of tree mappings although both are different.

Example 2.62 Consider a tree mapping $M = \{(s_5, t_4), (s_6, t_7), (s_{11}, t_{11}), (s_{13}, t_{12})\}$ between two trees S and T as shown in **Figure 2.26**. Now we focus on two nodes $s_2, s_{10} \in S$. For the subtree rooted at $s_2 \in S$, two nodes are included in $M^{(1)}$, i.e.

$$V(S(s_2)) \cap M^{(1)} = \{s_5, s_6\}.$$

Therefore, the root image of s_2 under M is

$$\mathbf{R}_M(s_2) = M(s_5) \sim M(s_6) = t_4 \sim t_7 = t_2.$$

For s_{10} , we have $\mathbf{R}_M(s_{10}) = t_{10}$. The shaded subtrees $S(s_2)$ and $S(s_{10})$ are isolated from each other, and the images of these subtrees under M , $T(t_2)$ and $T(t_{10})$, are also isolated. In the same way, we can confirm that, for any two nodes $x_1, x_2 \in S$, if the complete subtrees rooted at x_1 and x_2 are isolated from each other, then the images of these subtrees under M are also isolated. Hence, M is a structure-preserving mapping.

On the other hand, consider a tree mapping $M' = \{(s_5, t_4), (s_6, t_9), (s_{11}, t_{11}), (s_{13}, t_{12})\}$ between two trees

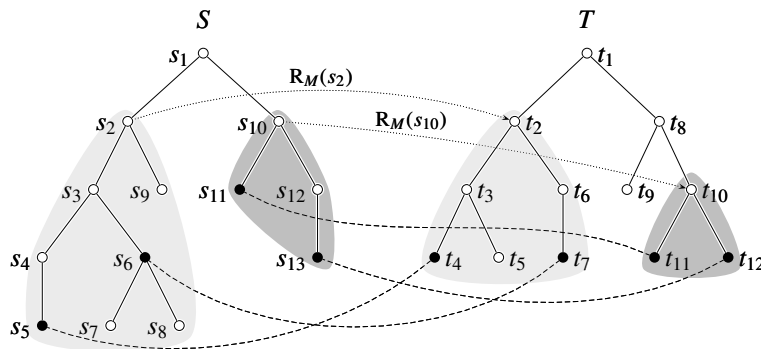


Figure 2.26. Structure-preserving mapping

S and T as shown in **Figure 2.27**. Two isolated subtrees $S(s_2)$ and $S(s_{10})$ have the overlapped images $T(t_1)$ and $T(t_{10})$ under M' . Hence, M' is not a structure-preserving mapping. ■

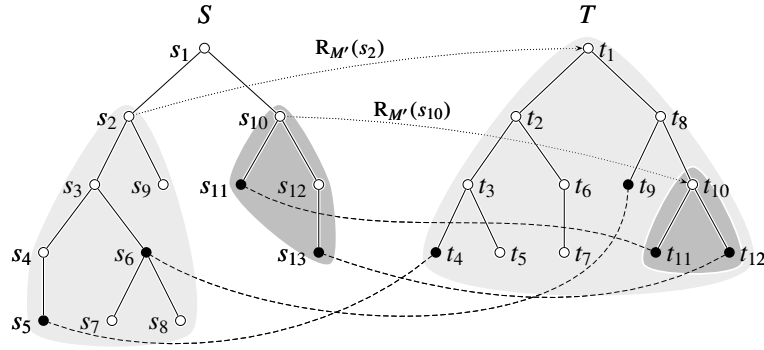
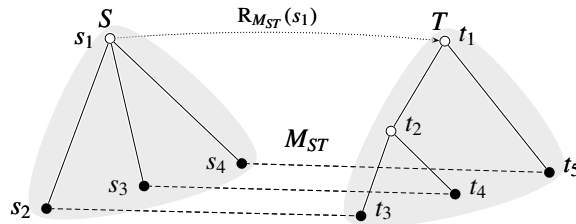


Figure 2.27. Non-structure-preserving mapping

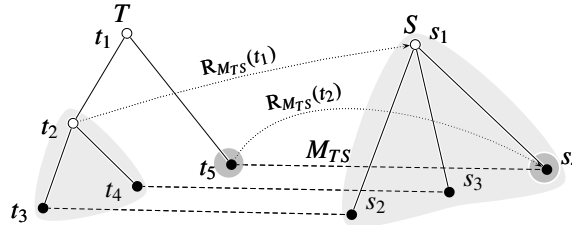
Proposition 2.63 (Asymmetry of Structure-Preserving Mapping)

The structure-preserving mapping is asymmetric.

Proof. Consider a tree mapping M_{ST} from S to T as shown in **Figure 2.28(a)**. It is easy to confirm that M_{ST} is a structure-preserving mapping from S to T . On the other hand, consider a tree mapping M_{TS} from



(a) Structure-preserving mapping from S to T



(b) Non structure-preserving mapping from T to S

Figure 2.28. Asymmetry of structure-preserving mapping

T to S in Figure 2.28(b). Although two subtrees $T(t_2)$ and $T(t_5)$ are isolated from each other, the images of these two subtrees under M_{TS} are overlapped. Hence, the tree mapping M_{TS} is not a structure-preserving mapping. ■

It is straightforward from Proposition 2.63 that the structure-preserving distance is not a metric. Also, it is easy to see that the structure-preserving mapping is monotonic and transitive.

Tanaka and Tanaka proposed an algorithm for computing structure-preserving distance [TT88], which runs in $O(|T_1| \cdot |T_2| \cdot \text{leaves}(T_2))$ time and $O(|T_1| \cdot |T_2|)$ space based on the following recurrences.

🍃 **Structure-respecting mapping does not exactly correspond to the algorithm.** Although we do not go into detail, these recurrences do not exactly correspond to the definition of structure-respecting mapping in Definition 2.61. The symmetric version of this algorithm corresponds to Lu’s algorithm.

$$\begin{aligned}
D(\emptyset, \emptyset) &= 0 \\
D(\emptyset, v(F) \bullet F') &= D(\emptyset, F \bullet F') + d(\varepsilon, v) \\
D(F, \emptyset) &= \sum_{T \in F} D_T(T, \emptyset) \\
D_T(v(F), \emptyset) &= D(F, \emptyset) + d(v, \varepsilon) \\
D(T_1 \bullet F'_1, v_2(F_2) \bullet F'_2) &= \min \begin{cases} D(T_1 \bullet F'_1, F_2 \bullet F'_2) + d(\varepsilon, v_2) \\ D_T(T_1, \emptyset) + D(F'_1, v_2(F_2) \bullet F'_2) \\ D_T(T_1, v_2(F_2)) + D(F'_1, F'_2) \end{cases} \\
D_T(v_1(F_1), v_2(F_2)) &= \min \begin{cases} D(v_1(F_1), F_2) + d(\varepsilon, v_2) \\ D(F_1, v_2(F_2)) + d(v_1, \varepsilon) \\ D(F_1, F_2) + d(v_1, v_2) \end{cases}
\end{aligned}$$

2.8.2 Strongly Structure-Preserving Distance

As shown in Section 2.8.1, the structure-preserving mapping is asymmetric. Tanaka defined the symmetric version of the structure-preserving mapping, and termed it as the *strongly-preserving mapping*.

Let M be a tree mapping from S to T . For $y \in T$, we define the *root image* of y under M as follows:

$$\mathbf{R}_M^{-1}(y) = \text{lca}(\{M^{-1}(t) \in T \mid y \leq_T t\}).$$

For a node $t \in T$, we refer to the complete subtree rooted at $\mathbf{R}_M^{-1}(t)$ as the image of $T(t)$ under M . The strongly-preserving mapping is defined as follows.

Definition 2.64 (Strongly Structure-Preserving Mapping [Tan84])

For two ordered trees S and T , a Tai mapping M from S to T is the *strongly structure-preserving* if the following two conditions are satisfied:

1. $\forall s_1, s_2 \in S \ [\mathbf{R}_M(s_1) \neq \perp \wedge \mathbf{R}_M(s_2) \neq \perp \implies [rl(s_1) \triangleleft_S s_2 \iff rl(\mathbf{R}_M(s_1)) \triangleleft_T \mathbf{R}_M(s_2)]]$.
2. $\forall t_1, t_2 \in T \ [\mathbf{R}_M^{-1}(t_1) \neq \perp \wedge \mathbf{R}_M^{-1}(t_2) \neq \perp \implies [rl(t_1) \triangleleft_T t_2 \iff rl(\mathbf{R}_M^{-1}(t_1)) \triangleleft_S \mathbf{R}_M^{-1}(t_2)]]$.



Strongly structure-preserving mapping & Lu mapping. Tanaka and Ohmori *et al.* stated in [Tan84] and [OT88] that the strongly structure-preserving mapping is equivalent to Lu mapping in Section 2.8.3. In fact, the strongly structure-preserving mapping is equivalent to the constrained mapping due to Zhang [Zha95], and a superclass of Lu mapping. We prove the fact in Section 4.3.

2.8.3 Lu Distance

Although Lu distance is not primarily defined by tree mapping, we introduce it here since it is closely related to the other tree mapping based distance measures.

Shin-Yee Lu proposed an algorithm [Lu79] of tree edit distance with the same intention as Tai distance, and applied it to clustering of handwritten characters. As mentioned in [SZ97, Section 14.2.4], this algorithm does not compute Tai distance despite Lu's intention, but computes some other distance. We refer to the distance measure as *Lu distance*. Later, the same algorithm for unordered trees was independently proposed by Aoki *et al.* [AYO⁺03] for glycan structure matching.

The algorithm for computing Lu distance is based on the following recurrences.

$$\begin{aligned}
& D(\emptyset, \emptyset) = 0 \\
& D(F, \emptyset) = \sum_{T \in F} D_T(T, \emptyset), & D(\emptyset, F) = \sum_{T \in F} D_T(\emptyset, T) \\
& D_T(v(F), \emptyset) = D(F, \emptyset) + d(v, \varepsilon), & D_T(\emptyset, v(F)) = D(\emptyset, F) + d(\varepsilon, v) \\
& D_T(v_1(F_1), v_2(F_2)) = \\
& \min \begin{cases} D_T(\emptyset, v_2(F_2)) + \min_{T \in F_2} \{D_T(v_1(F_1), T) - D_T(\emptyset, T)\}, & \langle \varepsilon \mapsto v_2 \rangle \\ D_T(v_1(F_1), \emptyset) + \min_{T \in F_1} \{D_T(T, v_2(F_2)) - D_T(T, \emptyset)\}, & \langle v_1 \mapsto \varepsilon \rangle \\ D(F_1, F_2) + d(v_1, v_2), & \langle v_1 \mapsto v_2 \rangle \end{cases} \quad (2.7a) \\
& D(T_1 \bullet F_1, T_2 \bullet F_2) = \min \begin{cases} D(F_1, T_2 \bullet F_2) + D_T(T_1, \emptyset) \\ D(T_1 \bullet F_1, F_2) + D_T(\emptyset, T_2) \\ D(F_1, F_2) + D_T(T_1, T_2) \end{cases} \quad (2.7b)
\end{aligned}$$

In these recurrences, Eq.(2.7a) is the same as Eq.(2.4a) for alignment distance, and Eq.(2.7b) is the same as the recurrence for string edit distance.

These recurrences are computed in $O(|T_1| \cdot |T_2|)$ time and space by using dynamic programming.

- 🍃 **Structure-preserving mapping & Lu mapping.** Shasha and Zhang stated in [SZ97, Section 14.7] that the algorithm introduced by Tanaka and Tanaka [TT88], i.e. the algorithm for computing the structure-preserving distance, is the same as Lu's algorithm [Lu79]. In fact, both are different. It is obvious from the fact that the structure-preserving mapping is asymmetric, while Lu mapping is symmetric.
- 🍃 **Tree mapping for Lu distance.** The tree mapping condition for Lu distance has been unknown. We reveal it in Section 4.8.

2.8.4 Constrained Distance — Isolated-Subtree Distance

This distance measure is also known as *isolated-subtree distance* [WZ01]. Based on almost the same intention as the structure-preserving mapping due to Tanaka and Tanaka [TT88], Zhang proposed a *constrained mapping* (or *isolated-subtree mapping*)[†], and designed a quadratic-time algorithm for ordered trees [Zha95] and a polynomial-time algorithm even for unordered trees [Zha96]. The constrained mapping is succinctly and naturally defined as follows.

Definition 2.65 (Constrained Mapping [Zha95, Zha96])

A Tai mapping M is *constrained* if the following condition holds:

$$\forall (s_1, t_1), (s_2, t_2), (s_3, t_3) \in M [s_3 < s_1 \sim s_2 \iff t_3 < t_1 \sim t_2].$$

Note that from Definition 2.65, it is obvious that

$$(s_1, s_2), (t_1, t_2) \in M [s_1 < s_2 \iff t_1 < t_2]$$

for any constrained mapping M , i.e. the condition of Tai mapping for unordered trees is implied.

For a tree mapping M from S to T , let M_1 and M_2 be two arbitrary subsets of M . An implication of the constrained mapping is that if two subtrees $S(\text{lca}(M_1^{(1)}))$ and $S(\text{lca}(M_1^{(2)}))$ are disjoint, i.e. $V(S(\text{lca}(M_1^{(1)}))) \cap V(S(\text{lca}(M_1^{(2)}))) = \emptyset$, then $T(\text{lca}(M_2^{(1)}))$ and $T(\text{lca}(M_2^{(2)}))$ must be disjoint as well, and vice versa.

From the definition, it is obvious that the constrained mapping is symmetric, and monotonic.

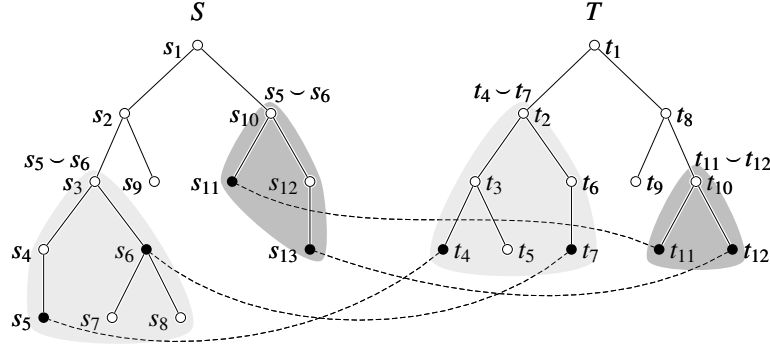


Figure 2.29. Constrained mapping

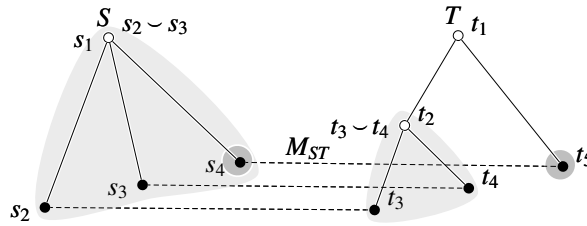


Figure 2.30. Non-constrained and structure-preserving mapping

Example 2.66 Consider the same example as Figure 2.26. It is easy to confirm that, by the tree mapping M , any disjoint two complete subtrees in S are mapped to disjoint two complete subtrees in T as shown in Figure 2.29, and vice versa.

On the other hand, the Tai mapping shown in Figure 2.10 is not constrained since $s_4 < s_6 \sim s_7$ and $t_3 \not< t_6 \sim t_9$.

The tree mapping in Figure 2.30 is not constrained while it is structure-preserving. Actually, it is easy to see that $s_4 < s_2 \sim s_3$ and $t_5 \not< t_3 \sim t_4$. ■

The constrained distance is proved to be a metric by the following proposition.

Proposition 2.67 (Transitivity of Constrained Mapping [Zha96, Lemma 2(1)])

The constrained mapping is transitive.

Proof. Let R , S and T be trees. Consider two constrained mappings $M_{RS} \in \mathcal{M}^{\text{CST}}(R, S)$ and $M_{ST} \in \mathcal{M}^{\text{CST}}(S, T)$. It follows from Proposition 2.48 that $M_{ST} \circ M_{RS}$ is a Tai mapping.

For each $i \in \{1, 2, 3\}$, let (r_i, t_i) be any element in $M_{RS} \circ M_{ST}$. Then, there exists s_i such that $(r_i, s_i) \in M_{RS}$ and $(s_i, t_i) \in M_{ST}$ for each $i \in \{1, 2, 3\}$. By the definition of constrained mappings, the following holds:

$$r_3 < r_1 \sim r_2 \Leftrightarrow s_3 < s_1 \sim s_2 \text{ and } s_3 < s_1 \sim s_2 \Leftrightarrow t_3 < t_1 \sim t_2.$$

Therefore, $r_3 < r_1 \sim r_2 \Leftrightarrow t_3 < t_1 \sim t_2$ holds. ■

Corollary 2.68 (Metric of Constrained Distance [Zha96, Theorem 2])

The constrained distance is a metric.

Proof. Since there exists an isomorphic mapping between the same two trees, and the constrained mapping is symmetric, then it is obvious that for any trees S and T ,

$$\mathbf{D}^{\text{CST}}(T, T) = 0 \text{ and } \mathbf{D}^{\text{CST}}(S, T) = \mathbf{D}^{\text{CST}}(T, S).$$

[†]Zhang refers to it as a *constrained edit distance mapping* in [Zha95, Zha96].

$$\begin{aligned}
& D(\emptyset, \emptyset) = 0 \\
& D(F, \emptyset) = \sum_{T \in F} D_T(T, \emptyset), & D(\emptyset, F) = \sum_{T \in F} D_T(\emptyset, T) \\
& D_T(v(F), \emptyset) = D(F, \emptyset) + d(v, \varepsilon), & D_T(\emptyset, v(F)) = D(\emptyset, F) + d(\varepsilon, v) \\
& D_T(v_1(F_1), v_2(F_2)) = \\
& \min \begin{cases} D_T(\emptyset, v_2(F_2)) + \min_{T \in F_2} \{D_T(v_1(F_1), T) - D_T(\emptyset, T)\}, & \langle \varepsilon \mapsto v_2 \rangle \\ D_T(v_1(F_1), \emptyset) + \min_{T \in F_1} \{D_T(T, v_2(F_2)) - D_T(T, \emptyset)\}, & \langle v_1 \mapsto \varepsilon \rangle \\ D(F_1, F_2) + d(v_1, v_2), & \langle v_1 \mapsto v_2 \rangle \end{cases} \quad (2.8a) \\
& D(F_1, F_2) = \min \begin{cases} D(\emptyset, F_2) + \min_{v(F'_2) \in F_2} \{D(F_1, F'_2) - D(\emptyset, F'_2)\} \\ D(F_1, \emptyset) + \min_{v(F'_1) \in F_1} \{D(F'_1, F_2) - D(F'_1, \emptyset)\} \\ D_S(F_1, F_2) \end{cases} \quad (2.8b) \\
& D_S(\emptyset, \emptyset) = 0 \\
& D_S(T \bullet F, \emptyset) = D_T(T, \emptyset) + D_S(F, \emptyset) \\
& D_S(\emptyset, T \bullet F) = D_T(\emptyset, T) + D_S(\emptyset, F) \\
& D_S(T_1 \bullet F_1, T_2 \bullet F_2) = \min \begin{cases} D_S(F_1, F_2) + D_T(T_1, T_2) \\ D_S(T_1 \bullet F_1, F_2) + D_T(\emptyset, T_2) \\ D_S(F_1, T_2 \bullet F_2) + D_T(T_1, \emptyset) \end{cases} \quad (2.8c)
\end{aligned}$$

Figure 2.31. Recurrences for computing constrained distance for ordered trees

Since tree mapping is subadditive by Lemma 2.40, and the constrained mapping is transitive by Proposition 2.67, it follows from Proposition 2.45 that the constrained distance is transitive, i.e. for any trees R , S , and T , we have

$$D^{\text{Cst}}(R, T) \leq D^{\text{Cst}}(R, S) + D^{\text{Cst}}(S, T).$$

Therefore, the constrained distance is a metric. ■

Zhang's Algorithm for Ordered Trees

Zhang proposed an efficient algorithm for computing the constrained distance for ordered trees [Zha95] based on the recurrences shown in **Figure 2.31**.

Note that an optimal constrained mapping is required (even if d is not a metric), Eq.(2.8a) should include the factor

$$D(F_1, F_2) + d(v_1, \varepsilon) + d(\varepsilon, v_2)$$

as in the case of Eq.(2.5) in alignment distance. The definition of $D_S(F_1, F_2)$ is the same as the string edit distance if two forests F_1 and F_2 are regarded as two strings consisting of trees instead of symbols. These recurrences are computed by using dynamic programming technique as shown in Algorithm 2.5.

Complexity. The computational complexity of this algorithm is estimated as follows. For each pair of nodes $v_1 \in T_1$ and $v_2 \in T_2$, the procedure `STRINGEDITDISTANCE` runs in $O(|\text{ch}(v_1)| \cdot |\text{ch}(v_2)|)$ time. The time complexity of computing $D(F_1, F_2)$ in Eq.(2.8b) and $D_T(T_1, T_2)$ in Eq.(2.8a) is bounded by $O(|\text{ch}(v_1)| + |\text{ch}(v_2)|)$. Hence, the factor of $O(|\text{ch}(v_1)| \cdot |\text{ch}(v_2)|)$ dominates the time complexity for each pair $v_1 \in T_1$ and $v_2 \in T_2$. Therefore, the time complexity of Algorithm 2.5 is

$$\begin{aligned}
\sum_{v_1 \in T_1} \sum_{v_2 \in T_2} O(|\text{ch}(v_1)| \cdot |\text{ch}(v_2)|) &\leq O\left(\sum_{v_1 \in T_1} |\text{ch}(v_1)| \times \sum_{v_2 \in T_2} |\text{ch}(v_2)|\right) \\
&\leq O(|T_1| \cdot |T_2|).
\end{aligned}$$

The space complexity is $O(|T_1| \cdot |T_2|)$ since we need to store $D_T(T_1(v_1), T_2(v_2))$ and $D(F_1(v_1), F_2(v_2))$ for each $v_1 \in T_1$ and $v_2 \in T_2$.

Algorithm 2.5 Constrained distance for ordered trees

```

procedure CONSTRAINEDDISTANCE( $T_1, T_2$ )
   $D(\emptyset, \emptyset) \leftarrow 0$ 
  foreach  $v \in T_1$  in postorder do
    let  $F_1$  be a forest such that  $T_1(v) = v(F_1)$ 
     $D(F_1, \emptyset) \leftarrow \sum_{T \in F_1} D_T(T, \emptyset)$ 
     $D_T(v(F_1), \emptyset) \leftarrow D(F_1, \emptyset) + d(v, \varepsilon)$ 
  foreach  $v \in V(T_2)$  in postorder do
    let  $F_2$  be a forest such that  $T_2(v) = v(F_2)$ 
     $D(\emptyset, F_2) \leftarrow \sum_{T \in F_2} D_T(\emptyset, T)$ 
     $D_T(\emptyset, v(F_2)) \leftarrow D(\emptyset, F_2) + d(\varepsilon, v)$ 

  for  $v_1 \in T_1$  in postorder do
    let  $F_1$  be a forest such that  $T_1(v_1) = v_1(F_1)$ 
    for  $v_2 \in T_2$  in postorder do
      let  $F_2$  be a forest such that  $T_2(v_2) = v_2(F_2)$ 
      
$$D(F_1, F_2) = \min \begin{cases} D(\emptyset, F_2) + \min_{v(F'_2) \in F_2} \{D(F_1, F'_2) - D(\emptyset, F'_2)\} \\ D(F_1, \emptyset) + \min_{v(F'_1) \in F_1} \{D(F'_1, F_2) - D(F'_1, \emptyset)\} \\ \text{STRINGEDITDISTANCE}(F_1, F_2) \end{cases}$$

      
$$D_T(v_1(F_1), v_2(F_2)) = \min \begin{cases} D_T(\emptyset, v_2(F_2)) + \min_{T \in F_2} \{D_T(v_1(F_1), T) - D_T(\emptyset, T)\} \\ D_T(v_1(F_1), \emptyset) + \min_{T \in F_1} \{D_T(T, v_2(F_2)) - D_T(T, \emptyset)\} \\ D(F_1, F_2) + d(v_1, v_2) \end{cases}$$

    return  $D_T(T_1, T_2)$ 
  end

procedure STRINGEDITDISTANCE( $T_1^1 \bullet \dots \bullet T_1^m, T_2^1 \bullet \dots \bullet T_2^n$ )
   $D_S[0, 0] \leftarrow 0$ 
  for  $i \leftarrow 1$  to  $m$  do
     $D_S[i, 0] \leftarrow D[i - 1, 0] + D_T(T_1^i, \emptyset)$ 
  for  $j \leftarrow 1$  to  $n$  do
     $D_S[0, j] \leftarrow D[0, j - 1] + D_T(\emptyset, T_2^j)$ 
  for  $i \leftarrow 1$  to  $m$  do
    for  $j \leftarrow 1$  to  $n$  do
      
$$D_S[i, j] \leftarrow \min \begin{cases} D_S[i - 1, j - 1] + D_T(T_1^i, T_2^j) \\ D_S[i - 1, j] + D_T(T_1^i, \emptyset) \\ D_S[i, j - 1] + D_T(\emptyset, T_2^j) \end{cases}$$

    return  $D_S[m, n]$ 
  end

```

Zhang's Algorithm for Unordered Trees

In contrast to Tai distance and the alignment distance problems, the constrained distance problem has a polynomial algorithm for unordered trees. Zhang proposed an algorithm for computing constrained distance for unordered trees [Zha96]. This algorithm has the same structure as that for ordered trees in Figure 2.31 except for Eq.(2.8c). Zhang designed the algorithm by replacing the string edit distance problem in Equation (2.8c) for ordered trees with a minimum cost maximum bipartite matching problem, and reducing it to the minimum cost maximum flow problem. This algorithm runs in $O(|T_1| \cdot |T_2| \cdot (\deg(T_1) + \deg(T_2)) \cdot \log_2(\deg(T_1) + \deg(T_2)))$ time for given two unordered trees T_1 and T_2 .

Improvements of Constrained Distance

Recently, Wang and Zhao proposed a more space efficient algorithm for computing constrained distance for ordered trees. It runs in $O((\log |T_1|) \cdot |T_2|)$ space with the same time complexity $O(|T_1| \cdot |T_2|)$ as Zhang's algorithm for two input trees T_1 and T_2 .

Ferraro and Godin improved the algorithm to find an optimal constrained mapping with minimum number of connected components [FG03] with the same computational complexity, and applied it to the plant comparison problem [FG00].


2.8.5 Structure-Respecting Distance

In order to design more structure-sensitive distance than Tai distance, Richter independently introduced the *structure-respecting mapping* [Ric97] for ordered trees with the same intention of the constrained mapping. The structure-respecting mapping is defined as follows.

Definition 2.69 (Structure-Respecting Mapping [Ric97]) A tree mapping M is *structure-respecting* if the following condition holds: for all $(s_1, t_1), (s_2, t_2), (s_3, t_3) \in M$ such that none of $s_1, s_2,$ and s_3 is an ancestor of the others,

$$s_1 \sim s_2 = s_1 \sim s_3 \iff t_1 \sim t_2 = t_1 \sim t_3.$$

From the definition of *structure-respecting distance*, Richter [Ric97] derived exactly the same algorithm as Zhang's algorithm for computing constrained distance for ordered trees [Zha95] in Algorithm 2.5. The difference between two algorithms arises just from the estimation of computational complexity. Zhang gave a tighter estimation than Richter. Richter estimates that this algorithm runs in $O(|T_1| \cdot |T_2| \cdot \deg(T_1) \cdot \deg(T_2))$ time and $O(\deg(T_1) \cdot \text{dep}(T_1) \cdot |T_2|)$ space (or $O(|T_1| \cdot |T_2|)$ space when tree mapping is required by traceback).

 **Constrained mapping & structure-respecting mapping.** The equivalence of two tree mapping classes defined in Definition 2.65 (constrained mapping) and 2.69 (structure-respecting mapping) has yet to be proved, although these two algorithms are the same, and Chin Lung Lu *et al.* stated that *both the concepts of constrained edit mapping and structure respecting mapping are equivalent*. We prove the equivalence between these two classes in Section 4.1.


2.8.6 Less-Constrained Distance

The *less-constrained mapping* was introduced by Chin Lung Lu *et al.* [LST01] with the intention of relaxing the condition of the constrained mapping [Zha95, Zha96] so that non-constrained mappings such as M_2 in **Figure 2.32(b)** are allowed while Tai mappings such as M_3 in **Figure 2.32(c)** remain prohibited. Lu *et al.* formulated the condition of less-constrained mapping as follows.

Definition 2.70 (Less-Constrained Mapping [LST01]) A Tai mapping M is *less-constrained* if the following condition holds: for all $(s_1, t_1), (s_2, t_2), (s_3, t_3) \in M$ such that none of $s_1, s_2,$ and s_3 is an ancestor of the others,

$$s_1 \sim s_2 \leq s_1 \sim s_3 \wedge s_1 \sim s_3 = s_2 \sim s_3 \iff t_1 \sim t_2 \leq t_1 \sim t_3 \wedge t_1 \sim t_3 = t_2 \sim t_3.$$

This definition is, in fact, critically inconsistent with the concept of less-constrained mapping as follows.

 **The concept of less-constrained mapping is not formulated by Definition 2.70.** We show an example that illustrates the inconsistency between the concept of less-constrained mapping and what Definition 2.70 implies.

Example 2.71 Consider the less-constrained mapping M_2 in **Figure 2.32(b)**. It is obvious that the following two conditions are satisfied in M_2 .

- $s_1 \sim s_2 \leq s_1 \sim s_3$ and $s_1 \sim s_3 = s_2 \sim s_3$.
- $t_1 \sim t_2 \not\leq t_1 \sim t_3$ and $t_1 \sim t_3 = t_2 \sim t_3$.

Therefore, the tree mapping M_2 is excluded by Definition 2.70. ■

Moreover, in Section 4.4, we prove that Definition 2.70 is mathematically equivalent to the definition of constrained mapping (Definition 2.65), and we give a correct definition.

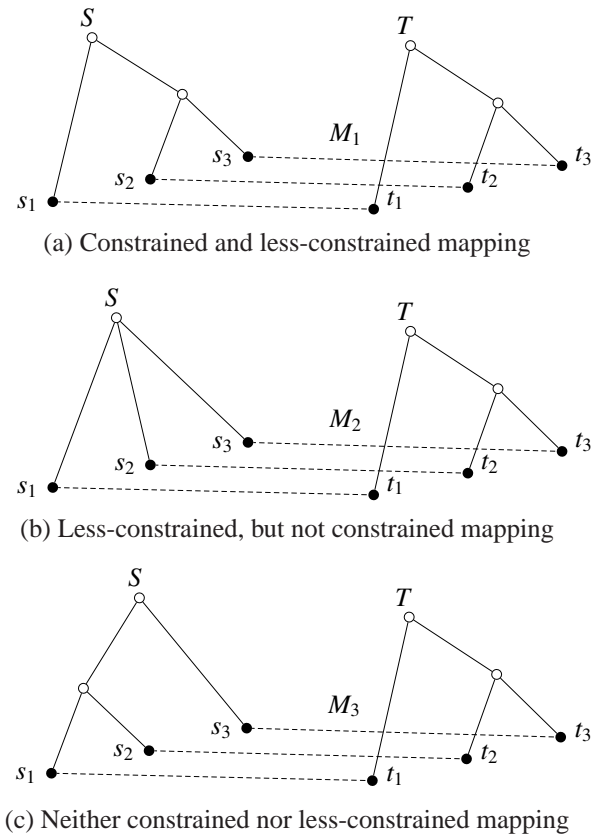


Figure 2.32. Feature of constrained and less-constrained mappings

The algorithm for computing less-constrained distance [LST01] for ordered trees was proposed based on the recurrences in **Figure 2.33**, where $D(F_1, F_2)$ denotes the alignment distance between two forests F_1 and F_2 , and $D_T(T_1, T_2)$ denotes the alignment distance between two trees T_1 and T_2 . While the mapping definition is inconsistent with the concept of less-constrained mapping, this algorithm is consistent with the concept of less-constrained mapping. These recurrences are the same as those for alignment distance except for Eq.(2.9c). This algorithm runs in $O(|T_1| \cdot |T_2| \cdot \deg(T_1)^3 \cdot \deg(T_2)^3 \cdot (\deg(T_1) + \deg(T_2)))$ time.

Less-constrained mapping & alignable mapping. We prove the equivalence between two classes of less-constrained mapping and the alignable mapping in Section 4.6.

Computational Complexity for Unordered Trees

Chin Lung Lu *et al.* showed that the less-constrained distance problem for unordered trees has no polynomial-time *absolute approximation* algorithm unless $P = NP$ [LST01]. A problem has an *absolute approximation* algorithm if, for any instance I of the optimization problem, the absolute error of the approximate solution $APP(I)$ is bounded by a constant c , i.e.

$$|APP(I) - OPT(I)| \leq c,$$

where $OPT(I)$ is an optimal solution of I (cf. [ACG⁺02, Section 3.1.1]). This result is more negative than MAX SNP-hard.

$$\begin{aligned}
& D(\emptyset, \emptyset) = 0 \\
& D(F, \emptyset) = \sum_{T \in F} D_T(T, \emptyset), & D(\emptyset, F) = \sum_{T \in F} D_T(\emptyset, T) \\
& D_T(v(F), \emptyset) = D(F, \emptyset) + d(v, \varepsilon), & D_T(\emptyset, v(F)) = D(\emptyset, F) + d(\varepsilon, v) \\
& D_T(v_1(F_1), v_2(F_2)) = \\
& \min \begin{cases} D_T(\emptyset, v_2(F_2)) + \min_{T \in F_2} \{D_T(v_1(F_1), T) - D_T(\emptyset, T)\}, & \langle \varepsilon \mapsto v_2 \rangle \\ D_T(v_1(F_1), \emptyset) + \min_{T \in F_1} \{D_T(T, v_2(F_2)) - D_T(T, \emptyset)\}, & \langle v_1 \mapsto \varepsilon \rangle \\ D(F_1, F_2) + d(v_1, v_2), & \langle v_1 \mapsto v_2 \rangle \end{cases} \quad (2.9a) \\
& D(F_1 \bullet T_1, F_2 \bullet T_2) = \min \begin{cases} D(F_1, F_2 \bullet T_2) + D_T(T_1, \emptyset) \\ D(F_1 \bullet T_1, F_2) + D_T(\emptyset, T_2) \\ D(F_1, F_2) + D_T(T_1, T_2) \\ D'(F_1 \bullet T_1, F_2 \bullet T_2) \end{cases} \quad (2.9b) \\
& D'(T_1^1 \bullet \dots \bullet T_1^m, T_2^1 \bullet \dots \bullet T_2^n) = \\
& \min \begin{cases} \min_{\substack{1 \leq i \leq k \leq m \\ 1 \leq j \leq n}} \left\{ \begin{aligned} & D(T_1^1 \bullet \dots \bullet T_1^{i-1}, T_2^1 \bullet \dots \bullet T_2^{j-1}) \\ & + D(T_1^i \bullet \dots \bullet T_1^k, F_2^j) + d(\varepsilon, v_2) \\ & + D(T_1^{k+1} \bullet \dots \bullet T_1^m, T_2^{j+1} \bullet \dots \bullet T_2^n) \end{aligned} \right\} \\ \min_{\substack{1 \leq i \leq m \\ 1 \leq j \leq k \leq n}} \left\{ \begin{aligned} & D(T_1^1 \bullet \dots \bullet T_1^{i-1}, T_2^1 \bullet \dots \bullet T_2^{j-1}) \\ & + D(F_1^i, T_2^j \bullet \dots \bullet T_2^k) + d(v_1, \varepsilon) \\ & + D(T_1^{i+1} \bullet \dots \bullet T_1^m, T_2^{k+1} \bullet \dots \bullet T_2^n) \end{aligned} \right\} \end{cases}, \quad (2.9c) \\
& \text{where let } T_1^i = v_1(F_1^i) \text{ and } T_2^j = v_2(F_2^j).
\end{aligned}$$

Figure 2.33. Recurrences for computing less-constrained distance for ordered trees

2.9 Subtree Isomorphism based Distance

The problem of determining whether two trees are isomorphic has been extensively studied and fundamental for a variety of tree pattern matching problems. In approximate tree matching, the problem of finding a common subtree[†] between two trees is an important generalization of tree isomorphism. This problem has a lot of variants, which are defined by tree mapping as well as the other tree edit distance measures.

2.9.1 Top-Down Distance — LCST Problem

The top-down distance is a tree edit distance measure with a simple restriction on edit operations. That is, applying deletions and insertions is confined to leaf nodes.

From the historical point of view, the edit-based approach to approximate tree matching probably dawned with the top-down distance due to Selkow [Sel77]. Later, Yang [Yan91] and Chawathe [Cha99a] applied the top-down distance to the change detection problem of two parse trees of programs toward a better alternative to the `diff` utility [HM76].

Wang and Zhang [WZ01] redefined the top-down distance by setting the *top-down mapping*.

Definition 2.72 (Top-Down Mapping [WZ01]) A Tai mapping M between two trees S and T is a *top-down mapping* if, for any pair $(s, t) \in M$ such that both s and t are non-root nodes, there exists also a pair $(\text{par}(s), \text{par}(t)) \in M$.

We show an example of top-down mapping in **Figure 2.36**. The algorithm for computing the top-down distance is based on the following operational definition [Sel77].

[†]The term *subtree* means a connected subtree pattern in a tree (See Remark 2.1).

$$\begin{aligned}
D(\emptyset, \emptyset) &= 0 \\
D(F, \emptyset) &= \sum_{T \in F} D_T(T, \emptyset), & D(\emptyset, F) &= \sum_{T \in F} D_T(\emptyset, T) \\
D_T(v(F), \emptyset) &= D(F, \emptyset) + d(v, \varepsilon), & D_T(\emptyset, v(F)) &= D(\emptyset, F) + d(\varepsilon, v) \\
D_T(v_1(F_1), v_2(F_2)) &= D(F_1, F_2) + d(v_1, v_2) \\
D(T_1 \bullet F_1, T_2 \bullet F_2) &= \min \begin{cases} D_T(T_1, T_2) + D(F_1, F_2) \\ D_T(T_1, \emptyset) + D(F_1, T_2 \bullet F_2) \\ D_T(\emptyset, T_2) + D(T_1 \bullet F_1, F_2) \end{cases}
\end{aligned}$$

In Algorithm 2.6, we show the algorithm proposed by Selkow [Sel77]. This algorithm runs in $O(|T_1| \cdot |T_2|)$ time and space.

Complexity. For two given trees T_1 and T_2 , the procedure `SUBTOPDOWNDISTANCE` is called once for each pair of nodes $v_1 \in T_1$ and $v_2 \in T_2$ with the same depth, i.e. $\text{dep}(v_1) = \text{dep}(v_2)$. In each call, `SUBTOPDOWNDISTANCE` runs in $O(|\text{ch}(v_1)| \cdot |\text{ch}(v_2)|)$. Therefore, the time complexity of Algorithm 2.6 is $O(|T_1| \cdot |T_2|)$ as in the case of the constrained distance problem for ordered trees.

Algorithm 2.6 Selkow's algorithm for top-down distance

```

procedure TOPDOWNDISTANCE( $T_1, T_2$ )
  /* precompute the global arrays  $D(\cdot, \cdot)$  and  $D_T(\cdot, \cdot)$  */
   $D(\emptyset, \emptyset) \leftarrow 0$ 
  foreach  $v \in T_1$  in postorder do
    let  $F_1$  be a forest such that  $T_1(v) = v(F_1)$ 
     $D(F_1, \emptyset) \leftarrow \sum_{T \in F_1} D_T(T, \emptyset)$ 
     $D_T(v(F_1), \emptyset) \leftarrow D(F_1, \emptyset) + d(v, \varepsilon)$ 
  foreach  $v \in V(T_2)$  in postorder do
    let  $F_2$  be a forest such that  $T_2(v) = v(F_2)$ 
     $D(\emptyset, F_2) \leftarrow \sum_{T \in F_2} D_T(\emptyset, T)$ 
     $D_T(\emptyset, v(F_2)) \leftarrow D(\emptyset, F_2) + d(\varepsilon, v)$ 
  return SubTopDownDistance( $T_1, T_2$ )
end

procedure SUBTOPDOWNDISTANCE( $v_1(T_1^1 \bullet \dots \bullet T_1^m), v_2(T_2^1 \bullet \dots \bullet T_2^n)$ )
   $D[0, 0] \leftarrow d(v_1, v_2)$ 
  for  $i \leftarrow 1$  to  $m$  do
     $D[i, 0] = D(T_1^1 \bullet \dots \bullet T_1^i, \emptyset)$ 
  for  $j \leftarrow 1$  to  $n$  do
     $D[0, j] = D(\emptyset, T_2^1 \bullet \dots \bullet T_2^j)$ 
  for  $i \leftarrow 1$  to  $m$  do
    for  $j \leftarrow 1$  to  $n$  do
       $D[i, j] = \min \begin{cases} D[i-1, j-1] + \text{SUBTOPDOWNDISTANCE}(T_1^i, T_2^j) \\ D[i-1, j] + D_T(T_1^i, \emptyset) \\ D[i, j-1] + D_T(\emptyset, T_2^j) \end{cases}$ 
  return  $D[m, n]$ 
end

```

Chawathe's Algorithm

All the algorithms for computing top-down edit distance due to Selkow [Sel77], Yang [Yan91] and Chawathe [Cha99a] run in $O(|T_1| \cdot |T_2|)$ time for given two trees T_1 and T_2 . The algorithms by Selkow and Yang include a recursive call as shown in Algorithm 2.6. On the other hand, Chawathe's Algorithm [Cha99a] is designed without recursive calls. In this algorithm, the top-down distance problem for two trees is reduced to the shortest path problem in a kind of lattice graph called an *edit graph*. The edit graph is constructed according to the following definition.

Definition 2.73 (Edit Graph for Two Trees) Let S and T be trees. Let $s_1, s_2, \dots, s_m \in S$ and $t_1, t_2, \dots, t_n \in T$ be sequences of nodes indexed by left-to-right preorder numbering, where $|S| = m$ and $|T| = n$. The *edit graph* of S and T is an edge weighted graph $G(S, T) = (V, E)$ such that

- the set of vertices V is $\{v_{(i,j)} \mid (i, j) \in \{0, \dots, m\} \times \{0, \dots, n\}\}$,
- the set of edges E consists of the following edges:
(Assume that two dummy nodes s_{m+1} and t_{n+1} with depth 0)
 - $\text{dep}(s_i) = \text{dep}(t_j) \iff (v_{(i-1,j-1)}, v_{(i,j)}) \in E$ with the weight $d(s_i, t_j)$ for $(i, j) \in \{1, \dots, m\} \times \{1, \dots, n\}$,
 - $\text{dep}(s_i) \geq \text{dep}(t_{j+1}) \iff (v_{(i-1,j)}, v_{(i,j)}) \in E$ with the weight $d(s_i, \epsilon)$ for $(i, j) \in \{1, \dots, m\} \times \{0, \dots, n\}$,
 - $\text{dep}(s_{i+1}) \leq \text{dep}(t_j) \iff (v_{(i,j-1)}, v_{(i,j)}) \in E$ with the weight $d(\epsilon, t_j)$ for $(i, j) \in \{0, \dots, m\} \times \{1, \dots, n\}$.

Given two trees S and T , each node of edit graph for S and T is represented as $D[i, j]$, which stores the top-down distance between two subforests $S[\{s_1, \dots, s_i\}]$ and $T[\{t_1, \dots, t_j\}]$ for $(i, j) \in \{0, \dots, |S|\} \times \{0, \dots, |T|\}$. We show Chawathe's algorithm in Algorithm 2.7.

Example 2.74 Consider two ordered trees in **Figure 2.34**, in which each node is indexed by left-to-right preorder numbering. **Figure 2.35(a)** shows the edit graph for two ordered trees S and T after computing $\text{TopDownDistance}(S, T)$, where all the edit costs are assumed to be 1 (unit cost). In **Figure 2.35(a)**, the distance (minimum weight) from top left to bottom right in the edit graph is computed, and it turns out to be 5 as given at the bottom right node. It is easy to design the traceback procedure for obtaining optimal top-down mappings as well as the procedure TRACEBACK in Algorithm 2.1 for strings. **Figure 2.1(b)** shows two shortest paths corresponding to two possible optimal top-down mappings in **Figure 2.36**. ■

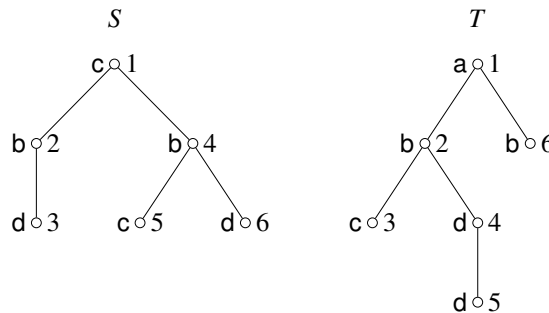


Figure 2.34. Two input trees in computing top-down distance: labels are attached to the left of the nodes, and sequential numbers in preorder to the right.

Largest Common Subtree (LCST) Problem for Unordered Trees

This problem is also known as *top-down maximum common subtree isomorphism* [Val98, Section 4.3]. The *largest common subtree* (LCST) problem is the top-down distance problem with LCS costs (See Section 2.6.2). Yang's algorithm [Yan91] has, in fact, the LCS-costing scheme.

This problem has been well-studied independently of the top-down distance problem, and some results have a significant impact across the board on the approximate tree matching problem.

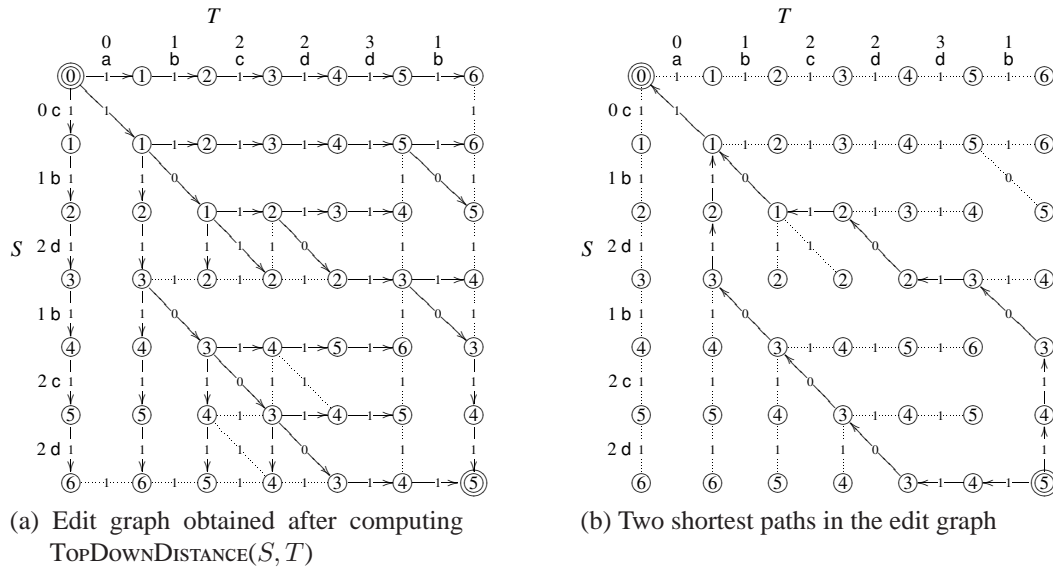


Figure 2.35. Edit graph for two ordered trees S and T

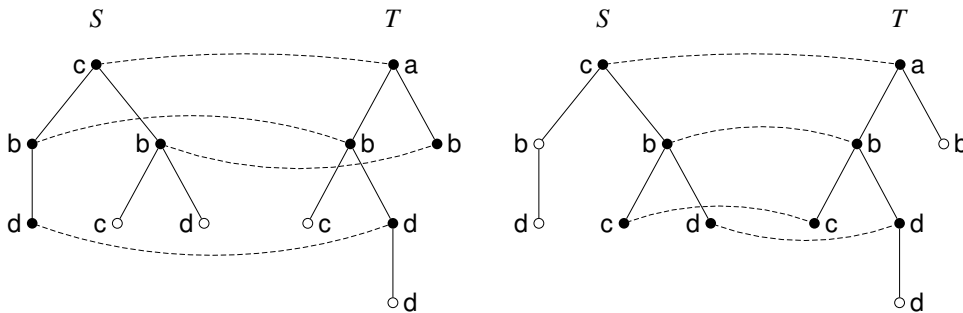


Figure 2.36. Top-down mappings

Algorithm 2.7 Chawathe's algorithm for top-down distance

```

procedure TopDownDistance( $S, T$ )
  Input: left-to-right preorder sequences of nodes
          $s_1, \dots, s_m \in S$ , where  $|S| = m$ 
          $t_1, \dots, t_n \in T$ , where  $|T| = n$ 
         let  $s_{m+1}$  and  $t_{n+1}$  be dummy nodes with depth 0
   $D[0, 0] \leftarrow 0$ 
  for  $i \leftarrow 1$  to  $m$  do
     $D[i, 0] \leftarrow D[i - 1, 0] + d(s_i, \epsilon)$ 
  for  $j \leftarrow 1$  to  $n$  do
     $D[0, j] \leftarrow D[0, j - 1] + d(\epsilon, t_j)$ 
  for  $i \leftarrow 1$  to  $m$  do
    for  $j \leftarrow 1$  to  $n$  do
       $x \leftarrow \infty; y \leftarrow \infty; z \leftarrow \infty$ 
      if  $\text{dep}(s_i) = \text{dep}(t_j)$  then  $x \leftarrow D[i - 1, j - 1] + d(s_i, t_j)$ 
      if  $\text{dep}(s_i) \geq \text{dep}(t_{j+1})$  then  $y \leftarrow D[i - 1, j] + d(s_i, \epsilon)$ 
      if  $\text{dep}(s_{i+1}) \leq \text{dep}(t_j)$  then  $z \leftarrow D[i, j - 1] + d(\epsilon, t_j)$ 
       $D[i, j] \leftarrow \min\{x, y, z\}$ 
  return  $D[m, n]$ 
end
  
```

There is an $O(n^3)$ -time algorithm for unordered trees, and an $O(n^2)$ -time algorithm for unlabeled unordered trees with bounded degree [Mat78]. Recently, Fukagawa and Akutsu proposed a fixed-parameter algorithm for unordered trees with a bounded size alphabet [FA06]. This algorithm runs in $O(4^k n)$ time for unordered trees, where k is a fixed upper bound of differences between two input trees T_1 and T_2 , and $n = \max\{|T_1|, |T_2|\}$. Even for unrooted trees, it runs in $O(4^k kn)$ time.

For more than two input unordered trees, Akutsu [Aku92] showed that the LCST problem is NP-hard, and moreover, Akutsu and Balledósson [AH00] showed that the LCST problem is also very hard to approximate. This result implies that the intractability of *multiple tree matching problems* based on not exclusively the LCST problem but also the other approximate tree matching problems.

2.9.2 Bottom-Up Distance

The bottom-up distance [Val01] is a tree edit distance measure with the restriction that applying deletions and insertions is confined to maximal nodes with respect to hierarchical order.

Valiente introduced the *bottom-up mapping* for formulating the bottom-up distance.

Definition 2.75 (Bottom-up mapping [Val01]) A Tai mapping M between two trees S and T is a *bottom-up mapping* if and only if, for any pair $(s, t) \in M$, the following hold.

1. $\forall s \in S [s \in M^{(1)} \implies \forall s' \in S(s) [s' \in M^{(1)}]]$.
2. $\forall t \in T [t \in M^{(2)} \implies \forall t' \in T(t) [t' \in M^{(2)}]]$.

The bottom-up mapping between two trees is the common complete subforest between two trees if labels are ignored.

Valiente proposed an $O(|T_1| + |T_2|)$ -time and -space algorithm [Val01] for computing the bottom-up distance for both ordered and unordered trees. This algorithm takes advantage of the algorithms for the common subexpression problem [DST80, FSS90]. Then, it basically computes the LCS-cost bottom-up mapping between two trees.



The bottom-up mapping is not a subclass of isolated-subtree or constrained mapping. Definition 2.75 is different from the original definition of bottom-up mapping [Val01] in which the bottom-up mapping is required to be a subclass of isolated-subtree mapping. Since Valiente's algorithm, in fact, does not compute isolated-tree mappings, we omit the requirement of isolated-subtree mapping in our definition of bottom-up mapping.

Example 2.76 Figure 2.37 shows an optimal bottom-up mapping between trees S and T (we only depict the mapping between maximal nodes with lines). This example is drawn from [Val01, Figure 8]. From this example, it is easy to verify that the bottom-up mapping is not constrained, i.e. not an isolated-subtree mapping. In fact, although $s_2 < s_7 \sim s_9$ does not hold, $t_4 < t_8 \sim t_{12}$ holds. ■

2.10 Related Work

There have been various directions to study related to approximate tree matching. Most of improvements and refinements in approximate tree matching have been made by taking advantage of the results in the field of stringology such as fixed-parameter algorithms, low-distortion embeddings, non-linear gap penalty, local similarity, multiple alignment. These topics are all intriguing and significant for real-world applications also in trees. In this section, we give a cursory review on some of these topics.

2.10.1 Hierarchical View of Tree Mappings

Wang and Zhang [WZ01] established a hierarchy among several distance measures based on the notion of tree mappings. Although there was some confusion in understanding the relationship between the structure-preserving distance [TT88] and the constrained distance [Zha95], the following was proved.

$$\text{Tai} \supseteq \text{ALN} \supseteq \text{CST} \supseteq \text{TOP}.$$

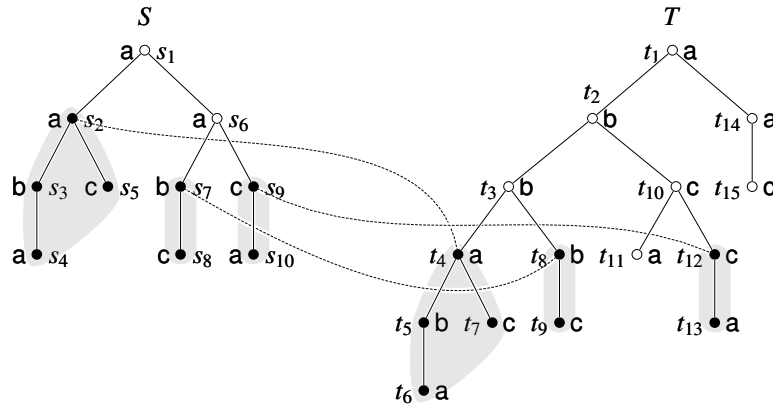


Figure 2.37. Bottom-up mapping

2.10.2 Tree Inclusion Problem

The tree inclusion problem is regarded a special case of approximate tree matching. For two trees P and T , P is *included* in T if P is obtained from T by deleting nodes in T .

The tree inclusion is also defined by a class of tree mappings. A tree mapping M from P to T is *left-total* if the following holds:

$$\forall p \in P [\exists t \in T \text{ such that } (p, t) \in M].$$

P is *included* in T if there exists a left-total Tai mapping M with LCS costs from P to T .

Since Kilpeläinen and Mannila proposed [KM95] a quadratic-time algorithm for ordered trees, some improvements have been made [Che98, Bil05].

The tree inclusion problem for unordered trees are proved to be NP-complete [MT92, KM95]. Valiente proposed a constrained tree inclusion [Val05] by confining deletions only to nodes with at most one child, and presented a polynomial time algorithm for unordered trees.

We summarize the complexities of these algorithms in **Table 2.8**.

Table 2.8. Computational complexity of tree inclusion problem

Trees	Class	Reference	Time	Space
ordered	general	[KM95]	$O(\ell_p n_t)$	$O(n_p n_t)$
	general	[Che98]	$O(\ell_p n_t)$	$O(\ell_p \min\{h_t, \ell_t\})$
	general	[BG05]	$O(\ell_p n_t)$	$O(n_p + n_t)$
	general	[BG05]	$O(n_p \ell_t \log \log n_t)$	$O(n_p + n_t)$
	general	[BG05]	$O(n_p n_t / \log n_t)$	$O(n_p + n_t)$
	constrained	[Val05]	$O(n_p n_t / \log n_t)$	$O(n_p n_t)$
unordered	general	[MT92, KM95]	NP-complete	
	constrained	[Val05]	$O(n_p^{1.5} n_t / \log n_t)$	$O(n_p n_t)$

$$n_p = |P|, \quad n_t = |T|, \quad \ell_p = |\text{leaves}(P)|, \quad \ell_t = |\text{leaves}(T)|, \quad h_t = \text{dep}(T)$$

2.10.3 Additional Edit Operations

In this thesis, we consider only the three elementary edit operations: replacements, deletions, and insertions. Barnard [BCD95] introduced *swapping operations* on two complete subtrees rooted at adjacent siblings into Zhang-Shasha's algorithm for computing Tai distance. Also, Bille [Bil03] incorporated *merge* and *split operations* on nodes into Zhang-Shasha's algorithm with some restrictions. Chawathe *et al.* [CRGMW96, CGM97, Cha99b] proposed efficient change detection algorithms including *move operations* on complete subtrees with heuristics for semi-structured data such as XML documents.

Magniez and Rougemont [MR07] proved that the Tai edit problem for ordered trees with move operations on complete subtrees is NP-complete even for binary trees by reducing the one-dimensional perfect BIN-PACKING problem into the Tai edit problem.

2.10.4 Gap Costs

Sakakibara [Sak03] proposed a polynomial-time algorithm for computing alignments of a specific type of binary trees with *affine gap penalties* for the analysis of RNA secondary structures. For more general gap penalties in tree edit distance, Touzet showed the following negative result.

Convex Gap Costs for Tai Distance

For a tree mapping M between two trees S and T , we refer to the set of (inserted or deleted) nodes $(V(S) \setminus M^{(1)}) \cup (V(T) \setminus M^{(2)})$ as the *gaps* in M . In this thesis, we consider only tree edit distance with *linear gap costs*, i.e. insertions and deletions are applied to nodes one by one, and the total cost is evaluated as the sum of each cost of editing one single node.

On the other hand, in string edit distance, a variety of costing schemes other than linear gap costs have been proposed in computational biology for confirming to biological models (cf. [Gus97, Section 11.8,12.6]). In analogy with the costing schemes in strings, Touzet [Tou03] proposed a new edit model called *gapped edit distance*. In this model, we can delete and insert a subtree with contiguous nodes by one step edit operation. A *convex gap cost* function is as a natural extension of it for strings, and defined as follows:

$$\text{cost}(\tau_1(\tau_2)) \leq \text{cost}(\tau_1) + \text{cost}(\tau_2),$$

where τ_1 and τ_2 are subtrees, and $\tau_1(\tau_2)$ is a subtree such that τ_2 is attached to a leaf of τ_1 . Touzet showed a negative result that the Tai edit problem with convex gap costs for ordered trees is NP-hard, while there exists a quadratic-time algorithm if gaps are restricted to complete subtrees [Tou03].

2.10.5 Local Similarity between Trees

If two large trees are not totally similar but share a small important subtree pattern, we need a specific method to find such a local pattern in stead of *global* tree matching methods. In strings, the algorithms for computing *local alignments* have been widely developed since Smith and Waterman first presented a local alignment algorithm [SW81]. The technique used in Smith and Waterman's algorithm is also applied straightforward to the algorithms for finding local similarity in tree mappings of the alignable and the constrained classes.

Höchsmann *et al.* [HTGK03] proposed an algorithm for computing an optimal local alignable mapping between ordered trees, and applied it to the analysis of RNA secondary structures [Höc05, Section 7.1]. This algorithm runs in $O(|T_1| \cdot |T_2| \cdot \deg(T_1) \cdot \deg(T_2) \cdot (\deg(T_1) + \deg(T_2)))$ time and $O(|T_1| \cdot |T_2| \cdot \deg(T_1) \cdot \deg(T_2))$ space for two input trees T_1 and T_2 . Ferraro and Ouangraoua [FO05] proposed an algorithm for computing an optimal local constrained mapping between unordered trees with the same complexity as Zhang's algorithm [Zha96].

Aoki *et al.* [AYO⁺03] developed a local approximate matching algorithm for unordered trees. This algorithm is a local matching and unordered tree version of Shin-Yee Lu's algorithm [Lu79].

2.10.6 Approximation of Tree Edit Distance

For a large data set of trees, as in the case of strings (Section 2.2.7), the computation of tree edit distance is often required to speed up even by sacrificing the accuracy of the computation.

In addition, if this metric space of tree edit distance can be embedded into a more familiar and tractable metric space such as Euclidean space while preserving the distances between each pair of trees, it may gain the understanding of the whole structure of given data.

Filtering by a Lower Bound

Inspired by *q-gram distance* for strings due to Ukkonen [Ukk92], a few similar methods have been proposed for trees.

Kailing *et al.* proposed distance measures [KKSS03] between unordered trees by combining a few very simple histograms of tree features; *i.e.*, degrees, heights, and labels of all nodes. These histogram-based distance measures are all computed in linear time, and are obviously lower bounds of Tai distance between unordered trees.

Augsten *et al.* [ABG05] introduced *pq-gram distance* between two ordered trees. The *pq-grams* are subtrees in which the number of leaves is q with the same depth p and the number of edges is $p + q - 1$. The *pq-gram distance* is defined based on the differences of the number of the occurrences of *pq-grams* between two input ordered trees. The *pq-gram distance* between two trees is computed in $O(n \log n)$ time and $O(n)$ space for the size of trees n . Although it is not shown that the *pq-gram distance* gives a theoretical lower bound of any tree edit distance, the 1,2-gram distance achieves empirically an effective approximation to Tai distance for real address data in the form of ordered trees.

Yang *et al.* [YKT05] also introduced the *binary branch distance* $\text{BBD}(T_1, T_2)$ between two ordered trees T_1 and T_2 . A *q-level binary branch* is a perfect binary tree[†] with depth q . Two input trees are normalized into binary tree representations. Then, the binary branch distance is defined based on the differences of the number of the occurrences of *q-level binary branches* between two input ordered trees. It is computed in $O(|T_1| + |T_2|)$ time. The lower bound is given as follows:

$$\text{BBD}(T_1, T_2) \leq (4 \cdot (q - 1) + 1) \cdot \mathbf{D}_1^{\text{Tai}}(T_1, T_2).$$

Tree Edit Distance Embeddings

As in the case of strings (Section 2.2.7), the low-distortion embedding problem for tree edit distance has very recently started to be addressed.

Garofalakis and Kumar [GK05] first introduced a low-distortion embedding algorithm of Tai distance for ordered trees with move operations on complete subtrees. This algorithm embeds a tree T into a vector image $\text{FV}(T)$ with at most $O(|T|)$ non-zero components in $O(|T| \log^* |T|)$ time. For two ordered trees T_1 and T_2 , and $n = \max\{|T_1|, |T_2|\}$, the following logarithmic distortion bounds are proved:

$$\mathbf{D}_1^{\text{Tai}}(T_1, T_2) \leq 5 \cdot \|\text{FV}(T_1) - \text{FV}(T_2)\|_1 = O(\log^2 n \log^* n) \cdot \mathbf{D}_1^{\text{Tai}}(T_1, T_2).$$

Moreover, Garofalakis and Kumar applied their algorithm to building a compact concise sketch synopses, and to approximating the similarity joins over streaming XML data.

Akutsu *et al.* [Aku06, AFT06] proposed two algorithms of embedding Tai distance for ordered trees into string edit distance. These embedding algorithms do not require move operations, but assume bounded degree trees. An input tree is coded into a string by using an Euler tour or a modified Euler tour (See [AFT06]). For a tree T , let us denote these two codings by $\text{EULER}(T)$ and $\text{EULER}'(T)$ respectively. Then the following hold.

$$\begin{aligned} \frac{1}{2h+1} \cdot \mathbf{D}_1^{\text{Tai}}(T_1, T_2) &\leq \mathbf{D}_1^{\text{Edit}}(\text{EULER}(T_1), \text{EULER}(T_2)) \leq 2 \cdot \mathbf{D}_1^{\text{Tai}}(T_1, T_2), \\ \frac{1}{O(n^{3/4})} \cdot \mathbf{D}_1^{\text{Tai}}(T_1, T_2) &\leq \mathbf{D}_1^{\text{Edit}}(\text{EULER}'(T_1), \text{EULER}'(T_2)) \leq 6 \cdot \mathbf{D}_1^{\text{Tai}}(T_1, T_2), \end{aligned}$$

where h is the minimum height of two input trees, and the sizes of both trees are assumed to be $O(n)$.

2.10.7 Edit Distance between Graphs

There has not been an established model of the edit distance for general graphs. Bunke *et al.* have recently given a couple of edit distance measures between two graphs based on the maximum common subgraph [Bun97], the maximal common subgraph [BS98], and the minimum common supergraph [BJK00]. On the other hand, Robles-Kelly and Hancock proposed methods for converting a graph into a string to exploit the established theory of string edit distance [RKH02, RKH03].

2.11 Summary

There are a lot of problem settings in approximate tree matching according to the factors considered such as tree types, costing schemes, edit operations, the classes of tree mappings. We mainly focus on labeled rooted trees, and formulate a variety of methods in a uniformed representation. Then we have clarified confusion and inconsistency in prior work as follows.

[†]a perfect binary tree is a tree in which every node has two or zero children, and all leaves are at the same depth.

-
- The tree mapping for alignment distance has not been known in any explicit formulation since Jiang and Wang proposed an algorithm for computing alignment of trees [JWZ95].
 - The formulation of less-constrained mapping due to Lu *et al.* [LST01] does not imply the tree mappings initially intended.
 - The relationship among structure-preserving, Lu, constrained, less-constrained, and structure-respecting distance measures are not clear, although some are regarded as the same.

The following two chapters are devoted to solve these problems.



Chapter 3

Theoretical Foundation of Approximate Tree Matching

In the previous chapter, we went through a variety of tree edit distance measures in both operational and declarative forms. Now we have found that there exist a few inconsistencies between declarative definitions and operational definitions.

In this chapter, we equip the notion of tree edit distance with an algebraic formulation for bridging the gap between operational and declarative semantics.

3.1 Preliminaries

We first show some important properties on interrelations of nodes in a tree with respect to least common ancestors.

Proposition 3.1 For any tree $T = (V, \leq)$, and any $x, y, z \in V$, the following properties hold.

1. For any subset of nodes $U \subseteq V$, there exists a unique least common ancestor of U .
2. $x \sim y$ and $x \sim z$ are comparable.
3. $x \sim x = x$.
4. $x \sim y = y \sim x$.
5. $(x \sim y) \sim z = x \sim (y \sim z)$.
6. $x \leq y \iff x \sim y = y$.
7. $y \leq x \wedge z \leq x \implies y \sim z \leq x$.
8. $x \leq x' \wedge y \leq y' \implies x \sim y \leq x' \sim y'$.
9. $\text{lca}(U) = \text{lca}(U \setminus W \cup \{\text{lca}(W)\})$ for non-empty W such that $W \subseteq U \subseteq V$.
10. $x \sim y < x \sim z \implies x \sim z = y \sim z$.
11. $x \sim y = x \sim z \implies y \sim z \leq x \sim y$.

Proof. 1. Let x and y be two least common ancestors of U . Then $x \leq y$ and $y \leq x$ hold. Hence $x = y$.

2. Both $x \leq x \sim y$ and $x \leq x \sim z$ hold. By Definition 2.17(2), $x \sim y$ and $x \sim z$ are comparable.

3–9. We omit these proofs since they are all obvious.

10. Since $y < x \sim z$ by the premise, we have $y \sim z \leq x \sim z$. We consider the following two cases since $x \sim y$ and $y \sim z$ are comparable by (2).

- Assume $x \sim y < y \sim z$, then $x < y \sim z$. We have $x \sim z \leq y \sim z$.
- Assume $x \sim y \geq y \sim z$, then $x \sim y \geq z$. We have $x \sim y \geq x \sim z$. It is contradictory to the premise.

Hence we have $y \sim z \leq x \sim z$ and $x \sim z \leq y \sim z$.

11. It follows from $z \leq x \sim z$ and the premise that $z \leq x \sim y$ holds. Moreover $y \leq x \sim y$ also holds, then $y \sim z \leq x \sim y$.

From Proposition 3.1.9–10, the relation among the least common ancestors of two nodes among given three nodes is summarized as follows.

Corollary 3.2 For any tree $T = (V, \leq)$, and any $x, y, z \in V$, any of the following properties holds.

1. $x \sim y < x \sim z$ and $x \sim z = y \sim z$.
2. $x \sim y = x \sim z$ and $y \sim z \leq x \sim z$.
3. $x \sim y > x \sim z$ and $x \sim y = y \sim z$.

For an ordered tree, there have several relations between the hierarchical order and the sibling order via least common ancestors.

Proposition 3.3 For any ordered tree $T = (V, \leq, \preceq)$, and any $x, y, z \in V$, the following properties hold.

1. $x \prec y \implies x < x \sim y \wedge y < x \sim y$.
2. $x \sim y < x \sim z \wedge x \prec z \implies y \prec z$.
3. $x \sim y < x \sim z \wedge x \succ z \implies y \succ z$.
4. $x \prec y \prec z \implies x \sim y \leq x \sim z$.
5. $x \prec y \prec z \implies y < x \sim z$.

Proof. 1. Since $x \prec y$, two nodes x and y are incomparable with respect to the hierarchical order, i.e. $x \not\leq y$ and $y \not\leq x$. We then immediately have $x < x \sim y$ and $y < x \sim y$.

2. Since $x \prec z$, we have $x \sim y \not\leq z$. Also we have $x \sim y \not\leq z$ since $x \sim y < x \sim z$. Hence $x \sim y$ and z are incomparable in the hierarchical order. It follows that $x \sim y \prec z$ because if $x \sim y \succ z$, we have $x \succ z$. Thus we obtain $y \prec z$ since $y \leq x \sim y$.

3. We omit this proof since this is symmetrical to (2).

4. If $x \sim y > x \sim z$ and $y \prec z$, then we have $y \prec x$ by (2). This is the contraposition of the assertion.

5. Since $x \prec y \prec z$, we have $y < x \sim y$ by (1), and $x \sim y \leq x \sim z$ by (4). Hence we have $y < x \sim z$.

3.2 Tree Homomorphism

In this section, we first introduce the notion of tree homomorphism. This notion stipulates a minimum requisite for a structure-preserving relationship between two trees. In the following two sections, By imposing further restrictions on the tree homomorphism, we provide two fundamental relationships between two trees.

Definition 3.4 (Tree Homomorphism) Let S and T be two unordered trees, an *unordered tree homomorphism*[†] from S to T is a (set-theoretic) mapping $f : V(S) \rightarrow V(T)$ that satisfies the following:

$$\forall x, y \in S [x \leq_S y \implies f(x) \leq_T f(y)].$$

When S and T are two ordered trees, we define an *ordered tree homomorphism* from S to T by adding the following condition to the unordered tree homomorphism.

$$\forall x, y \in S [x \prec_S y \implies f(y) \not\leq_T f(x)].$$

For an ordered or unordered tree homomorphism $f : V(S) \rightarrow V(T)$, we simply write $f : S \rightarrow T$. For a homomorphism $f : S \rightarrow T$, we define $f(V(S))$ as $\{f(x) \in T \mid x \in S\}$, and abuse the notation $f(S)$ to denote $f(V(S))$.

Proposition 3.5 For two ordered trees S and T , let $f : V(S) \rightarrow V(T)$ be a mapping that satisfies the following:

$$\forall x, y \in S [x \leq_S y \implies f(x) \leq_T f(y)].$$

Then the following two statements are equivalent.

1. $\forall x, y \in S [x \prec_S y \implies f(y) \not\prec_T f(x)]$.
2. $\forall x, y \in S [f(x) \prec_T f(y) \implies x \prec_S y]$.

Proof. (2 \Rightarrow 1): We now assume that $x \not\prec_S y$ for $x, y \in S$. Then one of $x \leq_S y$, $y \leq_S x$, or $y \prec_S x$ holds. It follows that

- $x \leq_S y \implies f(x) \leq_T f(y)$,
- $y \leq_S x \implies f(y) \leq_T f(x)$, or
- $y \prec_S x \implies f(x) \not\prec_T f(y)$.

For each case, we have $f(x) \not\prec_T f(y)$.

(1 \Rightarrow 2): We omit the proof since it is similar to the converse. ■

Example 3.6 For two ordered trees S and T , **Figure 3.1** depicts an ordered tree homomorphism f from S to T . Note that, in this example, $s_2 \prec_S s_3$ and $f(s_2) \not\prec_T f(s_3)$. This tree homomorphism is bijective but its inverse f^{-1} is not a tree homomorphism. ■

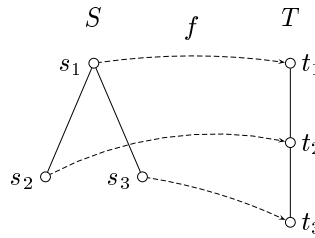


Figure 3.1. An ordered tree homomorphism f from S to T

Definition 3.7 (Tree Isomorphism) Let S and T be two ordered (resp. unordered) trees. An *ordered* (resp. *unordered*) tree isomorphism $f : S \rightarrow T$ is an ordered (resp. unordered) tree homomorphism such that the mapping f is bijective from $V(S)$ to $V(T)$, and the inverse mapping f^{-1} is also an ordered (resp. unordered) tree homomorphism.

We simply write a tree homomorphism and a tree isomorphism without “ordered” and “unordered” if the context is clear, or both ordered and unordered are considered in the context.

In graph theory, the notion of isomorphism is defined as follows:

Let $G = (V, E)$, and $G' = (V', E')$ be two graphs. Two graphs G and G' are *isomorphic*, denoted by $G \simeq G'$, if there exists a bijection (i.e. a one-to-one and onto mapping) $f : V \rightarrow V'$ such that

$$\forall x, y \in V [(x, y) \in E(G) \iff (f(x), f(y)) \in E(G')],$$

and such a mapping f is called an *isomorphism*.

We show that two unordered trees are isomorphic also as graphs if and only if there exists a tree isomorphism between them.

Proposition 3.8 Let S and T be two unordered trees. For a mapping $f : V(S) \rightarrow V(T)$, the following two statements are equivalent.

1. f is an unordered tree isomorphism from S to T .
2. f is a bijection such that $\forall x, y \in S [(x, y) \in E(S) \iff (f(x), f(y)) \in E(T)]$.

Proof. We only show that 1 implies 2 since the converse is straightforward. Define $g : V(S) \times V(S) \rightarrow V(T) \times V(T)$ by setting $g(x, y) = (f(x), f(y))$. Then it suffices to show $g(E(S)) \subseteq E(T)$.

For any edge $(x, y) \in E(S)$, let z be a node in $V(T)$ such that $f(x) \leq_T z <_T f(y)$. Since f^{-1} is a tree homomorphism, we have $x \leq_S f^{-1}(z) <_S y$, and hence $x = f^{-1}(z)$ because (x, y) is an edge of S . It follows from $z = f(x)$ that $(f(x), f(y))$ is an edge of T . ■

Proposition 3.9 Let S and T be two ordered trees. For a mapping $f : V(S) \rightarrow V(T)$, the following three statements are equivalent.

1. f is an ordered tree isomorphism from S to T .
2. f is an ordered tree homomorphism from S to T , and an unordered tree isomorphism from S to T .
3. f is a bijection and an unordered tree homomorphism from S to T such that

$$\forall x, y \in S [x \prec_S y \implies f(x) \prec_T f(y)].$$

Proof. (1 \Rightarrow 3): It is straightforward.

(3 \Rightarrow 2): It suffices to show that $x <_S y$ for any $x, y \in V(S)$ such that $f(x) <_T f(y)$. One of $x <_S y$, $x >_S y$, $x \prec_S y$ and $x \succ_S y$ holds, and we have $x \not\prec_S y$ and $x \not\succ_S y$ since we have $f(x) \prec_T f(y)$ or $f(x) \succ_T f(y)$ otherwise. Further, $x \not\prec_T y$ since f is a tree homomorphism of the unordered tree.

(2 \Rightarrow 1): If $f(x) \prec_T f(y)$, then $x \prec_S y$. We thus have f^{-1} is also a tree homomorphism. ■

Example 3.10 For unordered trees, the tree isomorphisms are not uniquely determined. **Figure 3.2** shows two tree isomorphisms from T to itself, i.e. there exists a non-trivial automorphism. ■

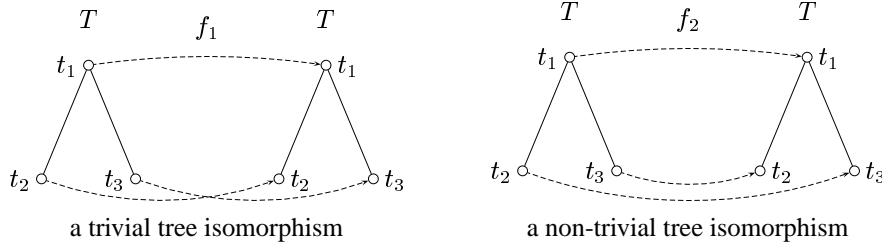


Figure 3.2. Tree isomorphisms for unordered trees.

In contrast to unordered trees, for two ordered trees S and T , a tree isomorphism from S to T is unique if it exists. This property follows the fact that there exists a unique *leftmost leaf* for any ordered tree T , where the leftmost leaf is defined as the node x of T such that

$$\nexists y \in T \text{ s.t. } (y < x) \quad \text{and} \quad \nexists y \in T \text{ s.t. } (y \prec x).$$

Hence if f is an ordered tree isomorphism from S to T , the leftmost node of S must be mapped to the leftmost node of T .

The following proposition can be proved by the induction on the size of two ordered trees S and T .

Proposition 3.11 Let S and T be two ordered trees. An ordered tree isomorphism from S to T is unique if it exists.

Corollary 3.12 Let T be an ordered tree. An ordered tree isomorphism $f : T \rightarrow T$ is identical to the identity map of $V(T)$

3.3 Tree Embedding

We introduce a variant of the tree homomorphism, *embedding*, which plays a central role to define the alignment of trees.

Definition 3.13 (Embedding) Let S and T be two ordered (resp. unordered) trees. An ordered (resp. unordered) *embedding* from S to T is an ordered (resp. unordered) tree homomorphism $f : S \rightarrow T$ such that

$$\forall x, y \in S [x \leq_S y \iff f(x) \leq_T f(y)].$$

We define the *residue* of f as $\text{res}(f) = |V(T(f(\text{root}(S)))) \setminus f(S)|$.

It is obvious from the definition that any embedding is an injection (i.e. a one-to-one mapping), and for any embedding $f : S \rightarrow T$, and any two node $x, y \in S$, we have $x = y \iff f(x) = f(y)$, and $x <_S y \iff f(x) <_T f(y)$.

Example 3.14 Figure 3.3 shows an embedding f from S to T . The residue of f is $\text{res}(f) = |V(T) \setminus f(S)| = |\{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8\} \setminus \{t_2, t_4, t_7\}| = 5$.

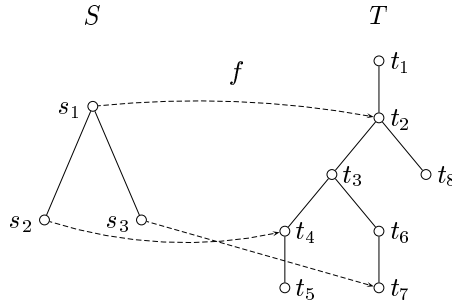
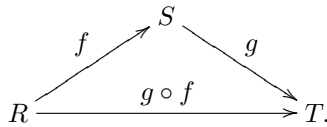


Figure 3.3. An ordered embedding f from S to T .

Proposition 3.15 Let S and T be ordered trees, and $f : S \rightarrow T$ be an ordered embedding. Then, for all $x, y \in S$, $f(x) \leq_T f(y)$ if $x \leq_S y$.

Proof. Assume that $f(x) \not\leq_T f(y)$. Then one of $f(y) <_T f(x)$, $f(x) <_T f(y)$, or $f(y) <_T f(x)$ holds. Thus one of $x < y$, $y < x$, or $y <_S x$ must hold. It follows that $x \not\leq_S y$.

Proposition 3.16 Let R , S , and T be (ordered or unordered) trees. Suppose that $f : R \rightarrow S$ and $g : S \rightarrow T$ are tree homomorphisms, i.e.



Then the following properties hold.

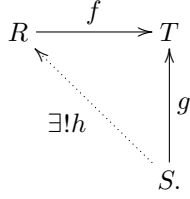
1. If f and $g|_{f(R)}$ are embeddings, the composite of f and g , i.e. $g \circ f$, is an embedding from R to T . Moreover, $\text{res}(g \circ f) = \text{res}(f) + \text{res}(g|_{f(R)})$ holds.
2. If $g \circ f$ is an embedding, then f is also an embedding.

Proof. 1. By Definition 3.13, for any $x, y \in S$, if $g(f(x)) \leq_T g(f(y))$ then $f(x) \leq_S f(y)$, and if $f(x) \leq_S f(y)$ then $x \leq_R y$. Therefore $g \circ f$ is an embedding. Also, $\text{res}(g \circ f) = |V(T)| - |(g \circ f)(R)| = (|V(S)| - |f(R)|) + (|V(T)| - |g|_{f(R)}(S)|) = \text{res}(f) + \text{res}(g|_{f(R)})$.

2. The mapping f is a tree homomorphism. Thus it suffices to show that $f(x) \leq_S f(y) \Rightarrow x \leq_R y$, for all $x, y \in R$. Now assume that $f(x) \leq_S f(y)$. Then we have $g(f(x)) \leq_T g(f(y))$ since g is a tree homomorphism. We obtain $x \leq_R y$ since the mapping $g \circ f$ is an embedding.

The embedding has a universal property in the following sense.

Proposition 3.17 (Universal Property of Embeddings) Let R , S , and T be three (ordered or unordered) trees. For an embedding $f : R \rightarrow T$, and a tree homomorphism $g : S \rightarrow T$, if $g(S) \subseteq f(R)$, then there exists a unique tree homomorphism $h : S \rightarrow R$ such that $g = f \circ h$, i.e.



Proof. Let us choose the unique (set-theoretic) mapping $h : V(S) \rightarrow V(R)$ so that $g = f \circ h$. We show that h is a tree homomorphism. Let x and y be two nodes in $V(S)$. From the definition of h , it follows that $g(x) = f(h(x))$ and $g(y) = f(h(y))$. Assume that $x \leq_S y$, then we have $g(x) \leq_T g(y)$. Since f is an embedding, we have $h(x) \leq_R h(y)$. ■

Corollary 3.18 Let R , S , and T be three trees, and $f : R \rightarrow T$ and $g : S \rightarrow T$ be two embeddings with $f(R) = g(S)$. There exists a unique isomorphism $h : S \rightarrow R$ such that $g = f \circ h$.

The following proposition and corollary show important relations between the embedding and the binary operator \smile , i.e. any embedding preserves the hierarchical order of least common ancestors between two trees.

Proposition 3.19 For an embedding $f : S \rightarrow T$, and any $x, y \in S$, let $z \in f(S) \subseteq V(T)$ be the minimum node with respect to the hierarchical order of T such that $f(x) \smile f(y) \leq_T z$. Then the node z is identical to $f(x \smile y)$. Furthermore $f(x) \smile f(y) <_T f(x \smile y)$ holds if and only if $f(x) \smile f(y) \notin f(S)$ holds. (See **Figure 3.4**.)

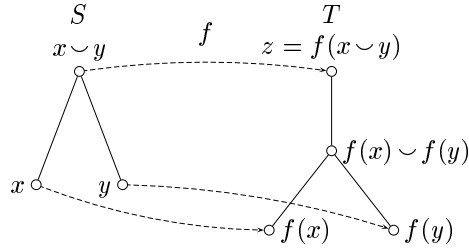


Figure 3.4. An example of the property in Proposition 3.19.

Proof. Let z be a node in $f(S) \subseteq V(T)$ such that $f(x) \smile f(y) \leq_T z$. Since f is an embedding, we have $x \smile y \leq_S f^{-1}(z)$. Hence $f(x \smile y) \leq_T z$. This implies that $f(x \smile y)$ is the minimum z such that $f(x) \smile f(y) \leq_T z$. The rest of the assertion is obvious. ■

Corollary 3.20 For any embedding $f : S \rightarrow T$, and any $x, y, z \in S$, if $x \smile y <_S x \smile z$, then $f(x) \smile f(y) <_T f(x) \smile f(z)$.

Proof. Since f is an embedding, by Proposition 3.19 we have $f(x) \smile f(y) \leq_T f(x \smile y) <_T f(x \smile z)$, and $f(x) \smile f(z) \leq_T f(x \smile z)$. If $f(x) \smile f(z) = f(x \smile z)$, then there is nothing to show. Here $f(x \smile y)$ and $f(x) \smile f(z)$ are comparable with respect to the hierarchical order of T . If there exists a node $w \in T$ such that $f(x) \smile f(z) \leq_T w <_T f(x \smile z)$, then w cannot be mapped by f from any node in $V(S)$ by Proposition 3.19, i.e. $w \neq f(x \smile y)$. Hence $f(x \smile y) <_T f(x) \smile f(z)$. Thus we conclude $f(x) \smile f(y) <_T f(x) \smile f(z)$. ■

Note that Corollary 3.20 cannot be extended to the 4-node case, i.e. for any embedding $f : S \rightarrow T$, even if $w \sim_S x \leq_S y \sim_S z$, the property $f(w) \sim_T f(x) \leq_T f(y) \sim_T f(z)$ does not necessarily hold. Actually **Figure 3.5** illustrates an example in which, for an embedding f , two nodes $f(s_1) \sim f(s_2)$ and $f(s_3) \sim f(s_4)$ are not comparable with respect to the hierarchical order in T , while $s_1 \sim_S s_2 <_S s_3 \sim_S s_4$ holds.

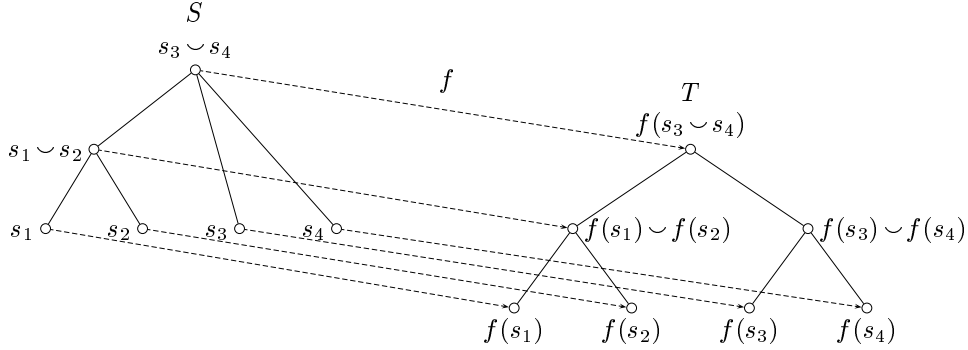


Figure 3.5. Corollary 3.20 cannot be extended to the 4-node case.

Proposition 3.21 Let T be a tree, and U be a subset of $V(T)$. If $T[U]$ is a tree, then the natural (set-theoretic) inclusion $f : U \rightarrow V(T)$ is an embedding, and $\text{res}(f) = |V(T) \setminus U|$.

We denote this embedding by $E_U : T[U] \rightarrow T$.

3.4 Insertion

Insertion of a node to a tree is known as a primitive operation used in tree edit distance. In what follows, we show the insertion is defined as a primitive variant of the embedding. More importantly, it is shown that any embedding is decomposed into one or more insertions.

Definition 3.22 (Insertion) Let S and T be two trees. An embedding $f : S \rightarrow T$ with $\text{res}(f) = 1$ is called an *insertion*. For an insertion $f : S \rightarrow T$ with a unique node $x \in V(T) \setminus f(S)$, the insertion f is said to be x -insertion into S , denoted by I_x .

Proposition 3.23 For any $x \in V(T) \setminus \{\text{root}(T)\}$, there exist a tree S and an x -insertion $I_x : S \rightarrow T$. Furthermore, the x -insertion is unique up to isomorphism.

Proof. Let x be a node in $V(T)$, and U be $V(T) \setminus \{x\}$. Then the natural inclusion $E_U : T[U] \rightarrow T$ is an x -insertion into $T[U]$ by Proposition 3.21. It follows from Corollary 3.18 that the x -insertion is unique up to isomorphism. ■

By the following theorem, we give the most important property of embeddings. That is, the embedding is identical to the composite of a series of insertions. In other words, any embedding corresponds to a transformation of a tree to another by repeatedly applying an insertion, i.e. a primitive edit operation.

Theorem 3.24 (Decomposition of Embedding) Let f be an embedding from S to T such that $V(T) \setminus f(S) = \{t_1, \dots, t_n\}$. There exists a series of trees T_0, T_1, \dots, T_n , and a series of insertions $f_i : T_i \rightarrow T_{i-1}$ for $i \in \{1, \dots, n\}$ such that

1. $T_0 = T$,
2. $T_n = S$,
3. $(f_1 \circ \dots \circ f_n)(T_n) = V(T_0) \setminus \{t_1, \dots, t_n\}$,

4. $f = f_1 \circ \dots \circ f_n$, i.e.

$$\begin{array}{ccccccc}
 T_n & \xrightarrow[I_{t_n}]{f_n} & T_{n-1} & \xrightarrow[I_{t_{n-1}}]{f_{n-1}} & \dots & \xrightarrow[I_{t_2}]{f_2} & T_1 \xrightarrow[I_{t_1}]{f_1} T_0 \\
 \parallel & & & & & & \parallel \\
 S & \xrightarrow{f} & & & & & T.
 \end{array}$$

Proof. We apply induction on $n = \text{res}(f)$. For $n = 1$, it is obvious that f is an insertion. Now we assume that $n \geq 2$. Let f_1 be the t_1 -insertion into T_1 . Note that f_1 can be naturally regarded as an insertion from T_1 to T_0 by Proposition 3.16.2. Now $f(T_n) \subseteq f_1(T_1)$ holds. Then there exists a unique tree homomorphism $g : T_n \rightarrow T_1$ such that $f = f_1 \circ g$ by Proposition 3.17. Moreover, by Proposition 3.16.2, g is also an embedding and $\text{res}(g) = n - 1$, and $g(T_n) = V(T_1) \setminus \{f_1^{-1}(t_2), \dots, f_1^{-1}(t_n)\}$ holds.

By the induction hypothesis, there exist a series of trees T_2, T_3, \dots, T_n , and a series of insertions $f_i : T_i \rightarrow T_{i-1}$ for $i \in \{2, \dots, n\}$ that satisfy the following:

1. $T_n = S$,
2. $(f_2 \circ \dots \circ f_i)(T_i) = V(T_1) \setminus \{f_1^{-1}(t_2), \dots, f_1^{-1}(t_i)\}$,
3. $g = f_2 \circ \dots \circ f_n$.

Then we have $(f_1 \circ \dots \circ f_i)(T_i) = V(T_0) \setminus \{t_1, \dots, t_i\}$ as follows:

$$\begin{aligned}
 (f_1 \circ \dots \circ f_i)(T_i) &= f_1(V(T_1) \setminus \{f_1^{-1}(t_2), \dots, f_1^{-1}(t_i)\}) \\
 &= f_1(T_1) \setminus \{t_2, \dots, t_i\} \\
 &= V(T_0) \setminus \{t_1, t_2, \dots, t_i\}.
 \end{aligned}$$

Then we complete the proof. ■

3.5 Tree Contraction

We introduce a variant of the tree homomorphism, *contraction* along with the embedding, which plays a central role to define the general tree edit distance due to Tai.

Definition 3.25 (Contraction) Let S and T be two (ordered or unordered) trees. A *contraction* from S to T is a tree homomorphism $f : S \rightarrow T$ such that

1. f is surjective onto $V(T)$,
2. $\forall x, y \in S [f(x) = f(y) \implies f(x \sim y) = f(x)]$,
3. $\forall x, y \in S [f(x) <_T f(y) \implies \exists z \in S \text{ s.t. } f(y) = f(z) \wedge x <_S z]$.

We define the *duplication* of f as $\text{dup}(f) = \{x \in S \mid f(x) = f(\text{par}(x))\}$.

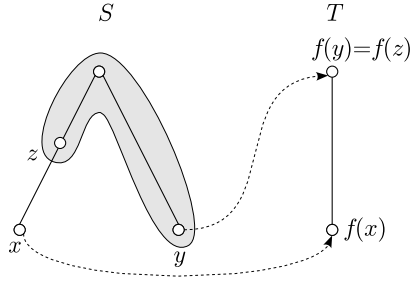


Figure 3.6. The definition of contraction

The second condition means that if two nodes are mapped to one node, then the least common ancestor of these two nodes is also mapped to the same node. The third condition with the second one means that if any incomparable two nodes x, y are mapped to comparable two nodes such that $f(x) <_T f(y)$, then all the nodes on the path between y and z ($z >_S x$) is also mapped to $f(y)$ (See **Figure 3.6**). Without the third condition, mappings such as shown in **Figure 3.1** are also allowed.

Example 3.26 **Figure 3.7** shows a contraction f from S to T . The duplication of f is $\text{dup}(f) = \{s_2, s_3, s_5, s_6, s_8\}$. **■**

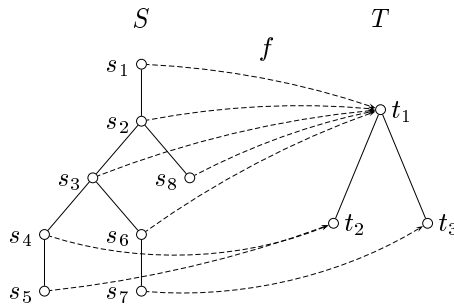


Figure 3.7. A contraction f from S to T

Proposition 3.27 Let S and T be two (ordered or unordered) trees. Any contraction $f : S \rightarrow T$ satisfies the following properties:

1. $f(\text{root}(S)) = \text{root}(T)$,
2. If $(x, y) \in E(S)$, then $(f(x), f(y)) \in E(T)$ or $f(x) = f(y)$,
3. For any node $x \in T$ and a node set $U = f^{-1}(x) \subseteq V(S)$, the least common ancestor of U is included in U , i.e. $\text{lca}(U) \in U$, and
4. $\text{dup}(f) = \bigcup_{x \in T} f^{-1}(x) \setminus \{\text{lca}(f^{-1}(x))\}$.

Proof. 1. It is straightforward since f is surjective and a tree homomorphism.

1. The case of $f(x) = f(y)$ is obvious. Then assume $f(x) <_T f(z) \leq_T f(y)$ for $x, y, z \in S$. By **Definition 3.25**, we may assume $x < z$. It follows from $(x, y) \in E(S)$ that $y \leq_T z$, and then $f(y) = f(z)$. This implies that $(f(x), f(y)) \in E(T)$.

2. For a node $x \in T$, let U be $f^{-1}(x)$. Choose $s \in U$ so that $s \not\leq_S y$ for all $y \in U$. We show that such $s \in U$ is identical to $\text{lca}(U)$. By **Definition 3.25**, for any node $y \in U$, we have $f(s \sim y) = x$, and hence $s \sim y = s$, i.e. $y \leq_S s$. Then $s = \text{lca}(U)$ holds.

3. Let t be a node in T . Assume that, for $x \in S$, $f(x) = t$ and $x \leq_S \text{lca}(f^{-1}(t))$. For all y such that $x <_S y \leq_S \text{lca}(f^{-1}(t))$, we have $f(y) = t$ since f is a tree homomorphism. In particular, $\text{par}(x) \in f^{-1}(t)$, and hence

$$\text{dup}(f) \supseteq \bigcup_{t \in T} f^{-1}(t) \setminus \text{lca}(f^{-1}(t)).$$

We have the converse since if $f(x) = f(\text{par}(x)) = t$, then $x \leq_S \text{lca}(f^{-1}(t))$, thus $\text{lca}(f^{-1}(t)) \notin \text{dup}(f)$. \blacksquare

The next proposition gives the very fundamental property that any contraction preserves the least common ancestors.

Proposition 3.28 (LCA-Preserving) Let S and T be two (ordered or unordered) trees. For any contraction $f : S \rightarrow T$ and $x, y \in S$, the following holds:

$$f(x \sim y) = f(x) \sim f(y).$$

The property is said to be *LCA-preserving*.

Proof. Since f is surjective onto $V(T)$, there exists $z \in S$ such that $f(z) = f(x) \sim f(y)$. Now we choose $x' \in S$ such that $x' \geq_S x$ and $f(x') = f(z)$. In the case of $f(x') <_T f(z)$, such x' exists since f is a contraction. On the other hand, in the case of $f(x') = f(z)$, it suffices to define $x' = x$. In the same way, we can choose $y' \in S$ such that $y' \geq_S y$ and $f(y') = f(z)$.

By the definition of the contraction, we have $f(x' \sim y') = f(z) = f(x) \sim f(y)$. Therefore $f(x \sim y) = f(x) \sim f(y)$ since $x' \sim y' \geq_S x \sim y$. \blacksquare

Corollary 3.29 Let $f : S \rightarrow T$ be a contraction. For any nodes $x, y, z \in S$ such that $x \sim y <_S x \sim z$, the following hold.

1. $f(x) \sim f(y) \leq_T f(x) \sim f(z)$,
2. $f(x) \sim f(y) = f(x) \sim f(z) \iff f(x \sim y) = f(x \sim z)$,
3. $f(y) \sim f(z) = f(x) \sim f(z)$.

Proof. Straightforward from Proposition 3.28. \blacksquare

Proposition 3.30 Let R , S , and T be three (unordered or ordered) trees. For two homomorphisms $f : R \rightarrow S$ and $g : S \rightarrow T$, the following properties hold:

1. If f and g are both contractions, then $g \circ f$ is also a contraction, $\text{dup}(g \circ f) = \text{dup}(f) \cup f^{-1}(\text{dup}(g))$,
2. If f is surjective onto $V(S)$ and $g \circ f$ is a contraction, then g is also a contraction.

Proof. 1. Since it is obvious that $g \circ f$ is surjective onto $V(T)$, we verify the remaining requirements for the contraction one by one.

First, we show $g(f(x \sim y)) = g(f(x))$ holds for any $x, y \in R$ such that $g(f(x)) = g(f(y))$. In fact, $g(f(x \sim y)) = g(f(x) \sim f(y)) = g(f(x)) \sim g(f(y)) = g(f(x))$ holds by Proposition 3.28.

Next, we show that there exists $z \in R$ such that $g(f(z)) = g(f(y))$ and $x <_R z$ for any $x, y \in R$ such that $g(f(x)) <_T g(f(y))$. Since g is a contraction, there exists $z' \in R$ such that $g(f(z')) = g(f(y))$ and $f(x) <_S f(z')$. In the same way, there exists $z \in R$ such that $f(z) = f(z')$ and $x <_R z$ since f is a contraction. Hence $g(f(z)) = g(f(z')) = g(f(y))$ holds.

The equation $\text{dup}(f \circ g) = \text{dup}(f) \cup f^{-1}(\text{dup}(g))$ is showed as follows. By Proposition 3.27, either $(f(x), g(y)) \in E(S)$ or $f(x) = g(y)$ holds for any $(x, y) \in E(R)$. Therefore, $g(f(x)) = g(f(y))$ holds if and only if either $x \in \text{dup}(f)$ or $f(x) \in \text{dup}(g)$ holds.

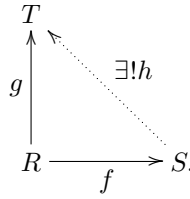
2. Since it is obvious that f is surjective onto $V(S)$, we verify the remaining requirements for the contraction one by one.

First, we show $g(f(x) \smile g(y)) = g(f(x))$ holds for any $x, y \in R$ such that $g(f(x)) = g(f(y))$. Note that $g(f(x)) \smile g(f(y)) \leq_T g(f(x) \smile f(y)) \leq_T g(f(x \smile y))$ generally holds. Since $g \circ f$ is a contraction, $g(f(x)) \smile g(f(y)) = g(f(x) \smile f(y)) = g(f(x \smile y))$ holds by Proposition 3.28. Hence, we obtain $g(f(x) \smile f(y)) = g(f(x))$.

Next, we show that there exists $z \in R$ such that $g(f(z)) = g(f(y))$ and $f(x) <_S f(z)$ for any $x, y \in R$ such that $g(f(x)) <_T g(f(y))$. Since $g \circ f$ is a contraction, there exists $z \in R$ such that $g(f(z)) = g(f(y))$ and $x <_R z$. Thus $f(x) <_S f(z)$ holds. \blacksquare

The contraction has a universal property in the following sense.

Proposition 3.31 (Universal Property of Contractions) Let R , S , and T be three (ordered or unordered) trees. For a contraction $f : R \rightarrow S$ and a tree homomorphism $g : R \rightarrow T$ such that $g(x) = g(y)$ if $f(x) = f(y)$ for any $x, y \in R$, there exists a unique homomorphism $h : S \rightarrow T$ satisfying $g = h \circ f$, i.e.



Proof. By the premise of the proposition, we have the unique set-theoretic mapping $h : S \rightarrow T$ such that $h \circ f = g$.

Next, we show that h is a tree homomorphism. Assume that $x, y \in R$ satisfy $f(x) < f(y)$. By assumption and Definition 3.25, we may assume $x <_R y$. Therefore, we have $g(x) \leq_T g(y)$. Hence h is an *unordered* tree homomorphism.

When f and g are ordered tree homomorphisms, so is h . In fact, if $g(x) \prec_T g(y)$, we have $x \prec_R y$. Therefore, one of $g(x) \prec_T g(y)$, $f(x) <_S f(y)$ and $f(x) >_S f(y)$ holds. If $f(x) <_S f(y)$ holds, we can choose y such that $x <_S y$ by assumption and Definition 3.25. Thus, this contradicts the assumption of $g(x) \prec_T g(y)$. In the same way, $f(x) >_S f(y)$ does not hold. \blacksquare

Corollary 3.32 Let R , S and T be three (ordered or unordered) trees. For $f : R \rightarrow S$ and $g : R \rightarrow T$ be both contractions such that, for any $x, y \in R$, $f(x) = f(y)$ if and only if $g(x) = g(y)$, there exists a unique isomorphism $h : S \rightarrow T$ such that $g = h \circ f$.

Proposition 3.33 Let T be an (ordered or unordered) tree, and U be a subset of $V(T)$ including the root of T . Assume that $T[U]$ is a tree, and let f be a surjective mapping from $V(T)$ to U such that $f(x) = x$ for all $x \in U$, and $f(x) = f(\text{par}(x))$ for all $x \notin U$. Then $f : V(T) \rightarrow U$ is a contraction with $\text{dup}(f) = V(T) \setminus U$.

We denote this contraction by $\mathcal{C}_U : T \rightarrow T[U]$.

Proof. First, we show that f is a tree homomorphism. Let x, y be two any nodes in $V(T)$. By definition, there exist $x', y' \in T$ such that $x \leq_T x'$, $f(x) = x'$, $y \leq_T y'$, and $f(y) = y'$. In other words, x' is the minimum ancestor of x such that $x' \in U$, and y' is the minimum ancestor of y such that $y' \in U$.

If $x \leq_T y$, then $x' \leq_T y'$. Therefore f is an unordered tree homomorphism. Also, for ordered T , if $x' \prec_T y'$, then $x \prec_T y$. Therefore f is an ordered tree homomorphism.

Finally, we verify that f satisfies the conditions for the contraction.

1. The mapping f is surjective by assumption.
2. If $x' = y'$, then $x \smile y \leq_T x' = y'$. Therefore, $f(x) \leq_T f(x \smile y) \leq_T f(x') = f(x)$ holds.
3. We have $x <_T y'$ since $x' <_T y'$.

We thus have the assertion. \blacksquare

3.6 Deletion

Deletion of a node, as well as insertion of a node, is a primitive edit operation to transform a tree to another. In this subsection, we formally define the primitive edit operation of node-deletion as a primitive variant of the tree homomorphism. At the same time, we present a property that characterizes the contraction from an operational point of view. We thus show that any contraction is identical to the composite of one or more deletion.

Definition 3.34 (Deletion) Let S and T be two (ordered or unordered) trees. A contraction $f : S \rightarrow T$ with $|\text{dup}(f)| = 1$ is called a *deletion*. In particular, f is called an x -deletion, if $\text{dup}(f) = \{x\}$, and denoted by D_x .

Proposition 3.35 For any $x \in S$ such that $x \neq \text{root}(S)$, there exist a tree T and an x -deletion $f : S \rightarrow T$. Furthermore, such an x -deletion is unique up to isomorphism.

Proof. Let U be $V(S) \setminus \{x\}$. Then $C_U : S \rightarrow S[U]$ is an x -deletion by Proposition 3.33. By Corollary 3.32, an x -degeneration is unique up to isomorphism. \blacksquare

In the same way as an embedding is decomposed into insertions, a contraction is decomposed into deletions.

Theorem 3.36 (Decomposition of Contractoin) Let f be a degeneration from S to T with $\text{dup}(f) = \{t_1, \dots, t_n\}$. There exist a series of trees T_0, T_1, \dots, T_n and a series of deletions $f_i : T_i \rightarrow T_{i+1}$ for $i \in \{0, \dots, n-1\}$ such that

1. $T_0 = S$,
2. $T_n = T$,
3. $\text{dup}(f_{i-1} \circ \dots \circ f_0) = \{t_1, \dots, t_i\}$,
4. $f = f_{n-1} \circ \dots \circ f_0$, i.e.

$$\begin{array}{ccccccc}
 T_0 & \xrightarrow[D_{t_0}]{f_0} & T_1 & \xrightarrow[D_{t_1}]{f_1} & \dots & \xrightarrow[D_{t_{n-2}}]{f_{n-2}} & T_{n-1} & \xrightarrow[D_{t_{n-1}}]{f_{n-1}} & T_n \\
 \parallel & & & & & & & & \parallel \\
 S & \xrightarrow{\quad\quad\quad f \quad\quad\quad} & & & & & & & T
 \end{array}$$

Proof. We prove the assertion by induction on n . When $n = 1$, f is a deletion by definition.

In the following, we assume $n \geq 2$. Let $f_0 : T \rightarrow T_1$ be D_{t_1} . By Proposition 3.31, there exists $g : T_1 \rightarrow T$ such that $f = g \circ f_0$. By Proposition 3.27, g is a contraction with $\text{dup}(g) = \text{dup}(f_0) \cup f_0^{-1}(\text{dup}(g))$.

Furthermore, $\text{dup}(g) = \{f_0(t_2), \dots, f_0(t_n)\}$ holds. In fact, if $f_0(t_1) \in \text{dup}(g)$, then $\text{par}(t_1) \in \text{dup}(f)$. Hence $f_0(t_1) \in \{f_0(t_2), \dots, f_0(t_n)\}$.

Thus, by applying the induction hypothesis to g , there exists a series of trees T_2, T_3, \dots, T_n as follows:

1. $T_n = T$,
2. $\text{dup}(f_{i-1} \circ \dots \circ f_1) = \{f_0(t_2), \dots, f_0(t_i)\}$ for $i \in \{2, \dots, n-1\}$, and
3. $g = f_{n-1} \circ \dots \circ f_1$.

Obviously $f = g \circ f_0 = f_{n-1} \circ \dots \circ f_0$ holds. In addition, the following holds.

$$\begin{aligned}
 \text{dup}(f_{i-1} \circ \dots \circ f_1 \circ f_0) &= \text{dup}(f_0) \cup f_0^{-1}(\text{dup}(f_{i-1} \circ \dots \circ f_1)) \\
 &= \{t_1, \dots, t_i\}
 \end{aligned}$$

We thus have the assertion. \blacksquare

3.7 Duality between Embedding and Contraction

The embedding and the contraction satisfy a certain duality property. In this subsection, we show the duality between embeddings and contractions.

Theorem 3.37 (Duality Theorem) For two (ordered or unordered) trees S and T , the following properties hold. (See **Figure 3.8**.)

1. For any contraction $f : S \rightarrow T$, there exists a unique embedding $g : T \rightarrow S$ such that $f \circ g$ is the identity map on $V(T)$ and $g \circ f$ is the identity map on $V(S) \setminus \text{dup}(g)$.
2. For any embedding $g : T \rightarrow S$ such that $g(\text{root}(T)) = \text{root}(S)$, there exists a unique contraction $f : S \rightarrow T$ such that $f \circ g$ is the identity map on $V(T)$ and $g \circ f$ is the identity map on $V(S) \setminus \text{dup}(f)$.

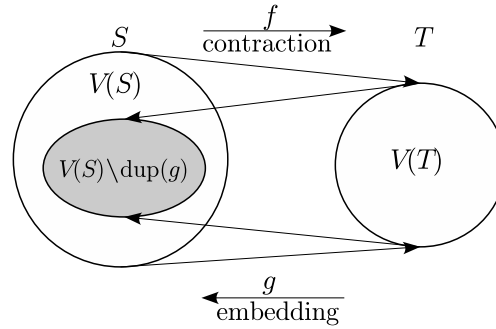


Figure 3.8. The duality between contraction f and embedding g

Proof. 1. Let U be $V(S) \setminus \text{dup}(f)$, and $h : S[U] \rightarrow S$ be the embedding defined by Proposition 3.21. Then $f \circ h$ is an isomorphism. In fact, since $f \circ h : S[U] \rightarrow T$ is a bijective tree homomorphism, it suffices to show that $x <_S y$ holds if $f(h(x)) <_T f(h(y))$ for any $x, y \in S$ according to Proposition 3.8.

By definition of the contraction, there exists $z \in V(S)$ such that $h(x) <_S z$ and $f(h(y)) = f(z)$. Since $h(y) = \text{lca}((f^{-1}(f(h(y))))))$, we have $h(x) <_S z \leq_S h(y)$. Hence $x <_S y$ since h is an embedding. Therefore, $g = h \circ (f \circ h)^{-1}$ is an embedding from T into S , and the rest of the assertion is obvious.

2. Let U be $g(T)$ and $h : S \rightarrow S[U]$ be the contraction defined by Proposition 3.33. In the same way as the proof to (1), it is shown that $h \circ g$ is an isomorphism. Therefore, $f = (h \circ g)^{-1} \circ h$ is a contraction from S onto T , and the rest of the assertion is obvious. ■

In the rest of this chapter and the next chapter, we use the following notation.

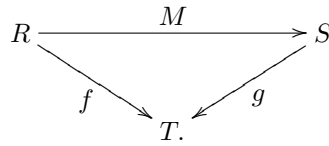
- By \bar{f} , we denote the embedding that is the dual of f in the sense of Theorem 3.37, provided $f : S \rightarrow T$ is a surjective contraction.
- By \bar{g} , we denote the contraction that is the dual of g in the sense of Theorem 3.37 provided $g : T \rightarrow S$ is an embedding such that $g(\text{root}(T)) = \text{root}(S)$.

Example 3.38 (Tai Mapping as a Common Subtree Pattern) We can define Tai mapping by using the notions introduced in this chapter. Recall that an alternative view of Tai mapping is a common subtree pattern shared in two trees. Then if we obtain two isomorphic trees by repeatedly deleting nodes from each tree, the corresponding nodes in the isomorphic two trees forms a Tai mapping. From this observation, we can give an alternative definition of Tai mapping as follows.

Definition 3.39 (Tai Mapping by Common Subtree Pattern) A tree mapping M from R to S is a *Tai mapping* if there exists a triplet (T, f, g) that satisfies the following.

1. $f : R \rightarrow T$ is a contraction.
2. $g : S \rightarrow T$ is a contraction.

$$3. M = \{\text{lca}(f^{-1}(x)), \text{lca}(g^{-1}(x)) \mid x \in T\}.$$



The triplet (T, f, g) is called a *common subtree pattern* between R and S on M .

3.8 Summary

Figure 3.9 illustrates our algebraic framework of approximate tree matching. In this chapter, we have established a theoretical foundation of edit-based approaches to approximate tree matching, which bridges the gap between operational semantics and declarative semantics of tree edit distance. This framework enables us to study mathematically the relationship among approximate tree matching methods defined in various ways.

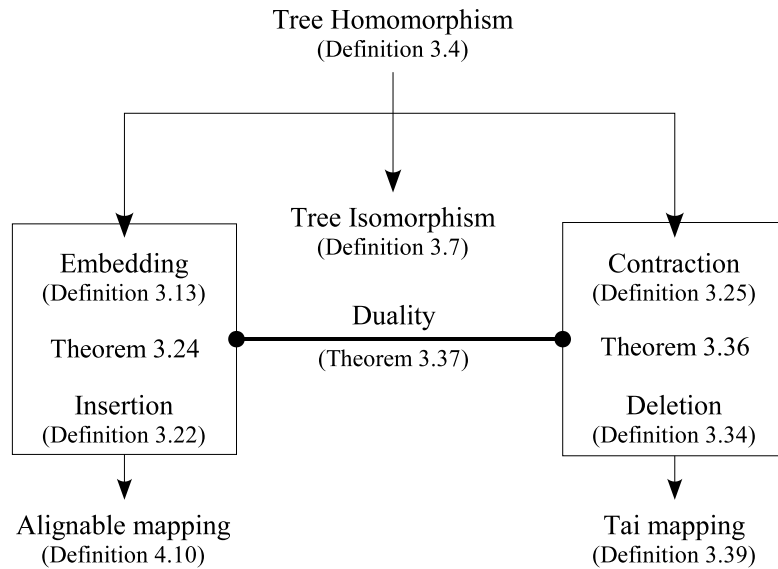
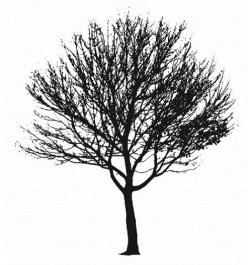


Figure 3.9. Algebraic formulation of approximate tree matching



Chapter 4

Relationship Analysis among Tree Edit Distance Measures

In this chapter, by comparing tree mapping conditions of a variety of tree edit distance measures, we reveal the relationship among them. First, we prove the equivalence between the constrained mapping and the structure-respecting mapping. Secondly, we show that the symmetric version of structure-preserving mapping is equivalent to the constrained mapping. In addition, we prove that the definition of less-constrained mapping due to Lu [LST01] turns out to be the definition of constrained mapping, and show the correct definition of less-constrained mapping.

Recall that the explicit definition of alignable mapping has remained unknown. We address this problem by using the algebraic framework developed in Chapter 2. Finally, we show some new facts on the classes of tree mappings, and summarize the hierarchy of tree mappings.

4.1 Constrained and Structure-Respecting Mappings: $Cst = SR$

We show the equivalence between structure-respecting and constrained mappings with a few other equivalent tree mappings.

Proposition 4.1 (Constrained Mapping & Structure-Respecting Mapping) For a Tai mapping M , the following are equivalent:

1. $\forall (s_1, t_1), (s_2, t_2), (s_3, t_3) \in M$,

$$s_1 \sim s_2 < s_1 \sim s_3 \wedge (\forall i, j \in \{1, 2, 3\} s_i \not\prec s_j) \iff t_1 \sim t_2 < t_1 \sim t_3 \wedge (\forall i, j \in \{1, 2, 3\} t_i \not\prec t_j).$$

2. $\forall (s_1, t_1), (s_2, t_2), (s_3, t_3), (s_4, t_4) \in M$,

$$s_1 \sim s_2 = s_3 \sim s_4 \wedge (\forall i, j \in \{1, \dots, 4\} s_i \not\prec s_j) \iff t_1 \sim t_2 = t_3 \sim t_4 \wedge (\forall i, j \in \{1, \dots, 4\} t_i \not\prec t_j).$$

3. For any $(s_1, t_1), (s_2, t_2), (s_3, t_3), (s_4, t_4) \in M$,

$$s_1 \sim s_2 < s_3 \sim s_4 \wedge (\forall i, j \in \{1, \dots, 4\} s_i \not\prec s_j) \iff t_1 \sim t_2 < t_3 \sim t_4 \wedge (\forall i, j \in \{1, \dots, 4\} t_i \not\prec t_j).$$

4. M is structure-respecting, i.e. $\forall (s_1, t_1), (s_2, t_2), (s_3, t_3) \in M$

$$s_1 \sim s_2 = s_1 \sim s_3 \wedge (\forall i, j \in \{1, 2, 3\} s_i \not\prec s_j) \iff t_1 \sim t_2 = t_1 \sim t_3 \wedge (\forall i, j \in \{1, 2, 3\} t_i \not\prec t_j).$$

5. M is constrained, i.e.

$$\forall (s_1, t_1), (s_2, t_2), (s_3, t_3) \in M [s_3 < s_1 \sim s_2 \iff t_3 < t_1 \sim t_2].$$

Proof. (1 \Rightarrow 4): We prove the contraposition of $s_1 \sim s_2 = s_1 \sim s_3 \implies t_1 \sim t_2 = t_1 \sim t_3$ under the condition that any of s_1, s_2, s_3 is not a proper ancestor of the others. If $t_1 \sim t_2 \neq t_1 \sim t_3$, we may assume $t_1 \sim t_2 < t_1 \sim t_3$ since $t_1 \sim t_2$ and $t_1 \sim t_3$ are comparable. $s_1 \sim s_2 < s_1 \sim s_3$ and therefore $s_1 \sim s_2 < s_1 \sim s_3$ immediately follows the condition (1).

(4 \Rightarrow 5): Assume that $s_3 < s_1 \sim s_2$. Note that, if t_3 and $t_1 \sim t_2$ are comparable, then $t_3 < t_1 \sim t_2$ holds. In fact, if $t_3 \geq t_1 \sim t_2$, then $s_3 \geq s_1$ and $s_3 \geq s_2$ holds, which contradicts the assumption of $s_3 < s_1 \sim s_2$. Hence, to show the assertion, it suffices to verify that t_3 is comparable with $t_1 \sim t_2$.

If some two of s_1, s_2, s_3 are comparable, s_3 is comparable with s_1 or s_2 (if $s_1 \leq s_2$ for example, $s_3 < s_1 \sim s_2 = s_2$ holds). Therefore, t_3 is comparable with $t_1 \sim t_2$ by the definition of the tree mapping.

Suppose that any two of s_1, s_2, s_3 are not comparable with each other. We may assume that $s_1 \sim s_2 = s_1 \sim s_3$ without loss of generality, and hence $t_1 \sim t_2 = t_1 \sim t_3$ since M is structure-respecting. In particular, t_3 is comparable with $t_1 \sim t_2$.

(5 \Rightarrow 1): Assume that $s_1 \sim s_2 < s_1 \sim s_3$ and any of s_1, s_2, s_3 is not a proper ancestor of the others (and therefore, any of t_1, t_2, t_3 is not a proper ancestor of the others).

Then, $t_3 \not\leq t_1 \sim t_2$ holds. In fact,

- (i) if $t_3 < t_1 \sim t_2$, then $s_3 < s_1 \sim s_2$ holds by (5);
- (ii) if $t_3 = t_1 \sim t_2$, either $s_3 = s_1$ or $s_3 = s_2$ holds since $t_3 \not\leq t_1$ and $t_3 \not\leq t_2$.

Any of the above is a contradiction to the assumption of $s_1 \sim s_2 < s_1 \sim s_3$. Hence, we conclude that $t_1 \sim t_2 < t_1 \sim t_3$.

(1 and 4 \Rightarrow 2 and 3): By Proposition 3.1, we assume $s_1 \sim s_3 = s_3 \sim s_4$ and therefore $t_1 \sim t_3 = t_3 \sim t_4$. $t_1 \sim t_2 < t_1 \sim t_3$ follows $s_1 \sim s_2 < s_1 \sim s_3$, and $t_1 \sim t_2 = t_1 \sim t_3$ does $s_1 \sim s_2 = s_1 \sim s_3$.

(2 \Rightarrow 4) and (3 \Rightarrow 1): To prove this, it suffices to let s_1 and s_3 be equal, and hence t_1 and t_3 be also equal. \blacksquare

Now we have proved that the constrained mapping is equivalent to the structure-respecting mapping, i.e.

$$\text{Cst} = \text{SR}.$$

4.2 Structure-Preserving and Constrained Mappings: $\text{SP} \supseteq \text{Cst}$

Before proving the relationship between the structure-Preserving mapping and the constrained mapping, we simplify the definition of structure-preserving mapping. From the definition of left-to-right preorder, the following holds for any nodes s_1 and s_2 in an ordered tree S (See **Figure 4.1**):

$$rl(s_1) \triangleleft_S s_2 \iff s_1 \prec_S s_2.$$

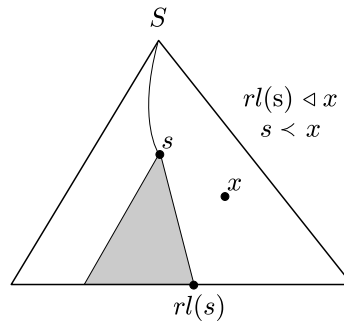


Figure 4.1. Left-to-right preorder and sibling order

Lemma 4.2 For a Tai mapping M from S to T , the following holds:

$$\forall s_1, s_2 \in S \left[\mathbf{R}_M(s_1) \neq \perp \wedge \mathbf{R}_M(s_2) \neq \perp \implies [\mathbf{R}_M(s_1) \prec_T \mathbf{R}_M(s_2) \implies s_1 \prec_S s_2] \right].$$

Proof. We consider only $s_1, s_2 \in S$ such that $R_M(s_1) \neq \perp$ and $R_M(s_2) \neq \perp$. Then, there exist $s'_1 \leq_S s_1$ and $s'_2 \leq_S s_2$ such that $s'_1, s'_2 \in M^{(1)}$. Further, we have $s'_1 \prec_S s'_2$ since $R_M(s_1) \prec_T R_M(s_2)$. This implies that $s_1 \succ_S s_2$ never happens. \blacksquare

Therefore, the definition of structure-preserving mapping in Definition 2.61 is simplified as follows.

Definition 4.3 (Structure-Preserving Mapping (2)) For two trees S and T , a Tai mapping M from S to T is structure-preserving if the following condition is satisfied:

$$\forall s_1, s_2 \in S \left[R_M(s_1) \neq \perp \wedge R_M(s_2) \neq \perp \implies [s_1 \prec_S s_2 \implies R_M(s_1) \prec_T R_M(s_2)] \right].$$

By using this, we prove the following proposition.

Proposition 4.4 For a Tai mapping M , if M is constrained, then M is structure-preserving, i.e.

$$SP \supseteq Cst.$$

Proof. Assume that M is a constrained mapping from S to T . We consider only nodes $s_1, s_2 \in S$ such that $R_M(s_1)$ and $R_M(s_2)$ are well-defined, i.e. $R_M(s_i) \neq \perp$ for $i \in \{1, 2\}$. Then, $R_M(s_1)$ and $R_M(s_2)$ respectively coincide with $y_1 \sim y_2$ and $y_3 \sim y_4$ such that $x_1 \sim x_2 \leq_S x_1$ and $x_3 \sim x_4 \leq_S x_2$ for some $(x_1, y_1), (x_2, y_2), (x_3, y_3) \in M$. The hypothesis $s_1 \prec_S s_2$ implies $x_1 \sim x_2 <_S x_1 \sim x_3$ and $x_3 \sim x_4 <_S x_1 \sim x_3$.

If none of x_i for $i \in \{1, \dots, 4\}$ is an ancestor of the others, $y_1 \sim y_2 <_T y_1 \sim y_3$ and $y_3 \sim y_4 <_T y_1 \sim y_3$ hold, since M is constrained. Therefore, $R_M(s_1) \prec_T R_M(s_2)$ immediately follows.

To complete the proof, It suffices to consider the following cases.

- (a) $x_1 \geq_S x_2$, and x_3 and x_4 are not comparable with respect to the hierarchical order.
- (b) $x_1 \geq_S x_2$ and $x_3 \geq_S x_4$.
- (c) Two nodes x_1 and x_2 are not comparable with respect to the hierarchical order, and $x_3 \geq_S x_4$.

Case (a): Any two of x_1, x_3 and x_4 are not comparable, and hence $y_3 \sim y_4 <_T y_1 \sim y_3$ holds, since M is constrained. Then $y_1 <_T y_1 \sim y_3$ follows from the hypothesis that M is a Tai mapping. Therefore, $R_M(s_1) = y_1 \prec_T y_3 \sim y_4 = R_M(s_2)$.

Case (b): Since $R_M(s_2) = y_3$ holds, the assertion immediately follows from the hypothesis that M is a Tai mapping.

Case (c): The proof is symmetrical to the proof for Case (a).

Then we have proved $SP \supseteq Cst$. \blacksquare

We have already seen $SP \neq Cst$ since the structure-preserving mapping is asymmetric while the constrained mapping is symmetric in Proposition 2.63. Hence, $SP \supsetneq Cst$ is concluded.

4.3 Strongly Structure-Preserving and Constrained Mappings: $SP^b = Cst$

The structure-preserving mapping is almost the same as the constrained mapping in its concept except for the asymmetry of structure-preserving mapping.

Here we prove that the symmetric version of structure-preserving mapping is equivalent to the constrained mapping. In the same way as the previous section, we simplify the definition of strongly structure-preserving mapping in Definition 2.64 as follows.

Definition 4.5 (Strongly Structure-Preserving Mapping (2)) For two ordered trees S and T , a Tai mapping M from S to T is *strongly structure-preserving* if the following two conditions are satisfied:

1. $\forall s_1, s_2 \in S \left[R_M(s_1) \neq \perp \wedge R_M(s_2) \neq \perp \implies [s_1 \prec_S s_2 \implies R_M(s_1) \prec_T R_M(s_2)] \right]$.
2. $\forall t_1, t_2 \in T \left[R_M(t_1) \neq \perp \wedge R_M(t_2) \neq \perp \implies [t_1 \prec_S t_2 \implies R_M(t_1) \prec_T R_M(t_2)] \right]$.

Proposition 4.6 For a Tai mapping M , M is constrained if and only if M is strongly structure-preserving, i.e.

$$\text{SP}^b = \text{Cst}.$$

Proof. From Proposition 4.4, $\text{SP}^b \supseteq \text{Cst}$ is obvious. Then we show that if M is strongly structure-preserving, then M is constrained, i.e. $\text{SP}^b \subseteq \text{Cst}$.

We here employ the following equivalent definition of constrained mapping (See Proposition 4.1):

$$\begin{aligned} &\forall (s_1, t_1), (s_2, t_2), (s_3, t_3) \in M \\ &[s_1 \sim s_2 < s_1 \sim s_3 \wedge (\forall i, j \in \{1, 2, 3\} s_i \not\prec s_j) \iff t_1 \sim t_2 < t_1 \sim t_3 \wedge (\forall i, j \in \{1, 2, 3\} t_i \not\prec t_j)]. \end{aligned}$$

Assume that M is a strongly structure-preserving mapping from S to T . Let $s_1 \sim s_2 <_S s_1 \sim s_3$ for $(s_1, t_1), (s_2, t_2), (s_3, t_3) \in M$ such that none of s_1, s_2 , and s_3 is an ancestor of the others. Without loss of generality, we may assume that $s_1 \prec_S s_2 \prec_S s_3$. Let x and x' be $s_1 \sim s_2$ and s_3 , respectively. Then, we have $x \prec_S x'$, and therefore, $R_M(s) \prec_T R_M(x')$ since M is strongly structure-preserving. It follows from $t_1 \sim t_2 \leq_T R_M(x)$ and $t_3 \leq_T R_M(x')$ that $t_1 \sim t_2 <_T t_1 \sim t_3$ holds. (Recall that if $x \prec y$, then $x < x \sim y$ and $y < x \sim y$ hold.)

In the same way, by assuming $t_1 \sim t_2 <_T t_1 \sim t_3$ for $(s_1, t_1), (s_2, t_2), (s_3, t_3) \in M$ such that none of s_1, s_2 , and s_3 is an ancestor of the others, we have $s_1 \sim s_2 <_S s_1 \sim s_3$.

Thus, M is constrained, i.e. $\text{SP}^b \subseteq \text{Cst}$. ■

4.4 Less-Constrained Mapping Revised

As mentioned in Section 2.8.6, Definition 2.70 due to Lu *et al.* [LST01] does not relax the condition of the constrained mapping. In fact, it is easy to show that the definition of less-constrained mapping due to Lu *et al.* is exactly equivalent to the definition of constrained mapping as follows.

Proposition 4.7 The condition for less-constrained mapping in Definition 2.70 is equivalent to the condition of the constrained mapping, i.e. the following two conditions are equivalent.

1. for all $(s_1, t_1), (s_2, t_2), (s_3, t_3) \in M$ such that none of s_1, s_2 , and s_3 is an ancestor of the others,

$$s_1 \sim s_2 \leq s_1 \sim s_3 \wedge s_1 \sim s_3 = s_2 \sim s_3 \iff t_1 \sim t_2 \leq t_1 \sim t_3 \wedge t_1 \sim t_3 = t_2 \sim t_3.$$

2. $\forall (s_1, t_1), (s_2, t_2), (s_3, t_3) \in M [s_3 < s_1 \sim s_2 \iff t_3 < t_1 \sim t_2]$.

Proof. Definition 2.70 is reduced to a more simple form of condition by Proposition 3.1(11); i.e., $x_i \sim y_i \leq x_i \sim z_i$ is implied by $x_i \sim z_i = y_i \sim z_i$ for $i \in \{1, 2\}$ in Definition 2.70. Therefore, it is shown that the condition due to Lu *et al.* is equivalent to that of the structure-respecting mapping in Definition 2.69. Hence, the condition due to Lu *et al.* is equivalent to the condition of constrained mapping. ■

We give a correct definition of the less-constrained mapping as follows.

Definition 4.8 ((Revised) Less-Constrained Mapping) A Tai mapping M is *less-constrained* if the following condition holds:

$$\forall (s_1, t_1), (s_2, t_2), (s_3, t_3) \in M [s_1 \sim s_2 < s_1 \sim s_3 \implies t_2 \sim t_3 = t_1 \sim t_3].$$

It is easy to confirm that this definition satisfies both cases in Figure 2.32(a) and (b). The revised definition of less-constrained mapping may seem somewhat incomplete since is asymmetric. However, by the following proposition, the symmetricity of the mapping condition is satisfied.

Proposition 4.9 The following two conditions are equivalent for a Tai mapping M .

1. $\forall (s_1, t_1), (s_2, t_2), (s_3, t_3) \in M [s_1 \sim s_2 < s_1 \sim s_3 \implies t_2 \sim t_3 = t_1 \sim t_3]$.
2. $\forall (s_1, t_1), (s_2, t_2), (s_3, t_3) \in M [t_1 \sim t_2 < t_1 \sim t_3 \implies s_2 \sim s_3 = s_1 \sim s_3]$.

Proof. By symmetry, it suffices to prove $(1 \Rightarrow 2)$. Then, by assuming the condition 1, we show that $t_1 \sim t_2 \not\prec t_1 \sim t_3$ if $s_2 \sim s_3 \neq s_1 \sim s_3$. Note that $s_2 \sim s_3$ and $s_1 \sim s_3$ are comparable.

- If $s_2 \sim s_3 < s_1 \sim s_3$, then we have $t_1 \sim t_2 = t_1 \sim t_3$ by the condition 1.
- If $s_2 \sim s_3 > s_1 \sim s_3$, then we have $t_1 \sim t_2 = t_2 \sim t_3$ by the condition 1. It follows that $t_1 \sim t_2 \leq t_1 \sim t_3$. Hence, $t_1 \sim t_3 \leq t_1 \sim t_2$.

Therefore, we have the condition 2. ■

4.5 Constrained and Less-Constrained Mappings: $\text{Cst}^\# \supseteq \text{Cst}$

It is obvious that the constrained mapping is not the less-Constrained mapping as shown in Figure 2.32(b). Then it suffices to prove the following proposition in order to show $\text{Cst}^\# \supseteq \text{Cst}$.

Proposition 4.10 A constrained tree mapping M is less-constrained, i.e.

$$\text{Cst}^\# \supseteq \text{Cst}.$$

Proof. Consider a constrained mapping M . We assume that $s_1 \sim s_2 < s_1 \sim s_3$ holds for $(s_1, t_1), (s_2, t_2), (s_3, t_3) \in M$, we prove that $t_2 \sim t_3 = t_1 \sim t_3$ holds. If none of s_i for $i \in \{1, 2, 3\}$ is an ancestor of the others, we have $t_1 \sim t_2 < t_1 \sim t_3$ since M is constrained. Therefore, $t_2 \sim t_3 = t_1 \sim t_3$ holds.

The remaining cases are as follows:

1. $s_1 < s_2$ or $s_2 < s_1$,
2. $s_1 < s_3$ and $s_2 < s_3$.

For the case (1), we can assume $s_1 < s_2$, and therefore $t_1 < t_2$, without loss of generality. Since s_2 is not an ancestor of s_3 , t_2 is not an ancestor of t_3 . Therefore, we have $t_1 \sim t_3 > t_2$ and $t_2 \sim t_3 \leq t_1 \sim t_3$ holds. The opposite inequality follows from $t_1 < t_2$.

In the case (2), $t_2 \sim t_3 = t_1 \sim t_3 = t_3$, since $t_1 < t_3$ and $t_2 < t_3$

Thus, we have shown that M is less-constrained. ■

4.6 Alignable and Less-Constrained Mapping: $\text{ALN} = \text{Cst}^\#$

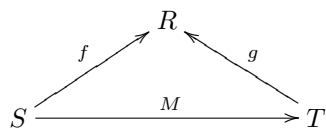
In this section, we prove that the alignable mapping is equivalent to the less-constrained mapping, i.e. $\text{ALN} = \text{Cst}^\#$. The definition of alignable mapping is, however, not explicitly known. Then we first formulate the alignable mapping by the algebraic framework developed in previous chapter before proving the equivalence.

4.6.1 Algebraic Formulation of Alignable Mappings

We study the property alignable mapping by using the algebraic framework introduced in Chapter 3. We redefine the alignable tree mapping as follows.

Definition 4.11 (Alignable Mapping) A tree mapping M from S to T is *alignable* if there exists a triplet (R, f, g) that satisfies the following.

1. $f : S \rightarrow R$ is an embedding.
2. $g : T \rightarrow R$ is an embedding.
3. $f(x) = g(y)$ for $(x, y) \in M$.



The triplet (R, f, g) is called an *aligned tree* of S and T on M .

Let us show several properties of alignable mappings.

Lemma 4.12 Note that M' be a subset of M . If M is alignable, then M' is also alignable.

Proof. An aligned tree on M is also an aligned tree on M' . ■

Lemma 4.13 For an alignable mapping M with an alignment tree (R, f, g) , $(s, t) = (s, \bar{g}(f(s)))$ holds for any $(s, t) \in M$, where \bar{g} is the contraction such that $\bar{g} \circ g$ is the identity map of T .

Proof. The assertion is obvious since $\bar{g} \circ g$ is the identity map of T . ■

Folding Two Sibling Subtrees. Let T be a rooted tree, v and w be two nodes in T such that $\text{par}(v) = \text{par}(w)$, and ν be a new node not in $V(T)$. We define a poset $T^{(vw)} = (V(T)^{(vw)}, \leq_T^{(vw)})$ as follows.

- $V(T)^{(vw)} = V(T) \setminus \{v, w\} \cup \{\nu\}$.
- For any two nodes $x, y \in V(T)^{(vw)}$, $x \leq_T^{(vw)} y$ holds if and only if any of the following holds.
 1. $x \neq \nu, y \neq \nu$ and $x \leq_T y$.
 2. $x = \nu$ and $y \geq_T v$ (i.e. $y \geq_T w$).
 3. $y = \nu$, and $x \leq_T v$ or $x \leq_T w$.

Figure 4.2 illustrates an example of the folding Two sibling subtrees.

Lemma 4.14 $T^{(vw)} = (V(T)^{(vw)}, \leq_T^{(vw)})$ is a rooted tree.

Proof. First, we show that $x \leq_T^{(vw)} z$ if $x \leq_T^{(vw)} y$ and $y \leq_T^{(vw)} z$.

- If $x, y, z \neq \nu$, then $x \leq_T y$ and $y \leq_T z$ hold, and therefore $x \leq_T z$ holds.
- If $x = \nu$, then $v \leq_T y$ and $y \leq_T z$ hold. Therefore $v \leq_T z$ holds.
- If $y = \nu$, then $x \leq_T v$ or $x \leq_T w$, and $z \geq_T v$ and $z \geq_T w$ hold. Therefore $x \leq_T z$ holds.
- If $z = \nu$, then $x \leq_T y$, and $y \leq_T v$ or $y \leq_T w$ hold. Therefore $x \leq_T v$ or $x \leq_T w$ holds.

Hence we have $x \leq_T^{(vw)} z$ for each case.

Secondly, we show that $(\uparrow x)_{T^{(vw)}} = \{y \in V(T)^{(vw)} \mid y \geq_T^{(vw)} x\}$ is a chain for every node x . Let us choose arbitrary two nodes $y, z \in (\uparrow x)_{T^{(vw)}}$. If $y \neq \nu$ and $z \neq \nu$, then $y \geq_T x$ and $z \geq_T x$ for $x \neq \nu$, or $y \geq_T v$ and $z \geq_T v$ for $x = \nu$ hold. In any case, $y \leq_T z$ or $y \geq_T z$ holds. Thus we have $y \leq_T^{(vw)} z$ or $y \geq_T^{(vw)} z$. If $y = \nu$, then $x \leq_T v$ or $x \leq_T w$, and $x \leq_T z$ hold. Therefore one of $v \leq_T z, w \leq_T z, z \leq_T v$ and $z \leq_T w$ holds. Hence we have $\nu \leq_T^{(vw)} z$ or $\nu \geq_T^{(vw)} z$. ■

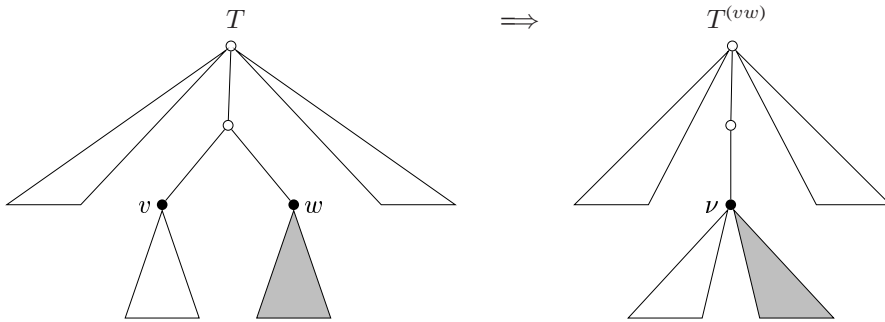


Figure 4.2. Folding two subtrees $T(v)$ and $T(w)$

Proposition 4.15 Let S and T be two trees. Any singleton tree mapping $M = \{(s, t)\}$ from S to T is alignable for any two nodes $s \in S$ and $t \in T$.

Proof. We start by constructing a tree $R' = (V(R'), \leq_{R'})$ from S and T as follows.

- $V(R') = V(S) \cup V(T)$.
- $x \leq_{R'} y$ holds for any two nodes $x, y \in R'$ if and only if any of the following holds.
 1. $x, y \in S$ and $x \leq_S y$.
 2. $x, y \in T$ and $x \leq_T y$.
 3. $x \in S$ and $y \in (\uparrow t)_T$.
 4. $x \in T(t)$ and $y \in (\uparrow s)_S$.

Note that $R' = (V(R'), \leq_{R'})$ is a rooted tree. We have $\text{par}(s) = \text{par}(t)$ in R' since $(\uparrow s)_{R'} = (\uparrow t)_{R'} = (\uparrow s)_S \cup (\uparrow t)_T$ by the definition of $\leq_{R'}$.

Thus, we can apply Lemma 4.14 to the tree R' , and $R = (V(R), \leq_R) = R'^{(s,t)}$ is a rooted tree. Moreover, it is easy to see that natural inclusion maps $f : V(S) \rightarrow V(R)$ and $g : V(T) \rightarrow V(R)$ are embeddings. In particular, since $f(s) = g(t)$ holds, $M = \{(s, t)\}$ is alignable. \blacksquare

Lemma 4.16 Let M be an alignable mapping from a tree S to a tree T . For two trees S and S' , let $h : S \rightarrow S'$ is an embedding. For a tree mapping M , the following are equivalent.

1. M is alignable.
2. $M' = \{(h(x), y) \mid (x, y) \in M\}$ is alignable.

Proof. (**2** \Rightarrow **1**) By the definition of the alignable mapping (Definition 4.11), it is obvious.

(**1** \Rightarrow **2**) Let (R, f, g) be an aligned tree on M . Then the embeddings $f : S \rightarrow R$ and $g : T \rightarrow R$ satisfy $f(s) = g(t)$ for all $(s, t) \in M$. By Theorem 3.24, it suffices to consider the following two cases.

1. h is not surjective and $\text{res}(h) = 0$.
2. $h(S) = S'$ and $\text{res}(h) = 1$.

Case 1: Letting $V(R') = V(S') \setminus h(S) \cup V(R)$, we define the relation $\leq_{R'}$ over $V(R')$ such that, for $x, y \in V(R')$, $x \leq_{R'} y$ if and only if one of the following holds.

- (i) $x, y \in V(S') \setminus h(S)$ and $x \leq_{S'} y$;
- (ii) $x, y \in R$ and $x \leq_R y$;
- (iii) $x \in R$ and $y \in (\uparrow h(\text{root}(S)))_{S'}$.

Note that $R' = (V(R'), \leq_{R'})$ is a tree. Let

$$\alpha : (V(S') \setminus h(S)) \rightarrow V(R') \quad \text{and} \quad \beta : V(R) \rightarrow V(R')$$

denote the natural inclusions. We define $f' : V(S') \rightarrow V(R')$ with

$$f'(x) = \begin{cases} \alpha(x) & \text{if } x \in V(S') \setminus h(S) \\ \beta(f(h^{-1}(x))) & \text{if } x \in h(S). \end{cases}$$

Also, we define $g' : V(T) \rightarrow V(R')$ with $g' = \beta \circ g$. It is easy to see that both f' and g' are embeddings. Since $f'(h(s)) = \beta(f(h^{-1}(h(s)))) = \beta(f(s)) = \beta(g(t)) = g'(t)$, we have the conclusion in the case of (1).

Case 2: In the following, we use the following notation.

- $v : V(S') \setminus h(S) = \{v\}$.
- $w : \text{par}(v) = h(w)$.
- $c_i : \text{ch}(v) = \{h(c_1), \dots, h(c_n)\}$.

Now, letting $V(R') = V(R) \cup \{v\}$, we define the relation $\leq_{R'}$ over $V(R')$ such that, for $x, y \in R'$, $x \leq_{R'} y$ if and only if one of the following holds.

1. $x, y \in R$ and $x \leq_{R'} y$.
2. $x = v'$ and $y \geq_{R'} f(w)$.
3. $x \in R$, and $\exists z \in R[f(c_i) \leq_{R'} z < f(w) \wedge x \leq_{R'} z]$ and $y = v'$.

Note that $R = (V(R'), \leq_{R'})$ is a tree

Letting $\alpha : V(R) \rightarrow V(R')$ be the natural inclusion, we define $f' : V(S') \rightarrow V(R')$ with $f'(x) = \alpha(f(h^{-1}(x)))$ if $x \neq v$ and $f'(v) = v'$. Further, we define $g' = \alpha \circ g : V(T) \rightarrow V(R')$. Note that both f' and g' are embeddings. Since $f'(h(s)) = \alpha(f(h^{-1}(h(s)))) = \alpha(f(s)) = \alpha(g(t)) = g'(t)$, we have the conclusion in the case of (2). \blacksquare

This lemma implies that if M is alignable after inserting nodes, it is also alignable before applying the insertions.

By the definition of tree mapping, without loss of generality, we may assume that for $(s, t) \in M$ if $s = \text{root}(S)$, then $t = \text{root}(T)$.

Lemma 4.17 Let (R, f, g) be an aligned tree on M . Then there exist f' and g' such that (R, f', g') is also an aligned tree on M and $f'(\text{root}(S)) = g'(\text{root}(T))$. In particular, the following are equivalent.

1. M is alignable.
2. $M \cup \{\text{root}(S), \text{root}(T)\}$ is alignable.

Proof. Let $(s, t) \in M$. Two nodes $f(\text{root}(S))$ and $g(\text{root}(T))$ are comparable in R since they are ancestors of $f(s) = g(t)$. If $f(\text{root}(S)) = g(\text{root}(T))$, there is nothing to prove.

Without loss of generality, we may assume that $f(\text{root}(S)) < g(\text{root}(T))$. Define $f' : V(S) \rightarrow V(R)$ with $f'(x) = f(x)$ if $x \neq \text{root}(S)$ and $f'(\text{root}(S)) = g(\text{root}(T))$. In the following, we show that f' is an embedding. First, assume that $x \leq_S y$ for $x, y \in S$. If $y \neq \text{root}(S)$, then $f'(x) <_R f'(y)$ holds since f is a tree homomorphism. If $y = \text{root}(S)$, then $f'(x) = f(x) <_R f(\text{root}(S)) <_R f'(\text{root}(S))$ holds. Thus f' is a tree homomorphism. The property $x < y$ if $f'(x) < f'(y)$ is also easily proved. Consequently, we see that f' is an embedding.

Since the (2) \Rightarrow (1) follows from Lemma 4.12, it suffices to show (1) \Rightarrow (2). As shown in the first part, if (R, f, g) , we have another aligned tree (R, f', g') such that $f'(\text{root}(S)) = g'(\text{root}(T))$. Therefore, $M \cup \{\text{root}(S), \text{root}(T)\}$ is alignable. \blacksquare

4.6.2 Equivalence between Alignable and Less-Constrained Mappings

In this section, we prove that the alignable mapping is equivalent to the less-constrained mapping.

In the following lemma, we prove that an alignable mapping can be constructed from tree mappings between subtrees.

Lemma 4.18 Let M be a tree mapping from a tree S to a tree T . Let S_i and T_i be as follows:

- S_i : the tree $S(s_i)$ for $\text{ch}(\text{root}(S)) = \{s_1, \dots, s_m\}$, and
- T_i : the tree $T(t_i)$ for $\text{ch}(\text{root}(T)) = \{t_1, \dots, t_n\}$.

By symmetry we assume that $m \leq n$. By $M_i \subset V(S_i) \times V(T_i)$ for $i \in \{1, \dots, m\}$, we denote the tree mapping $\{(s, t) \in M \mid s \in S_i \text{ and } t \in T_i\}$. If $M = \bigcup_{i=1}^m M_i$ and each M_i for $i \in \{1, \dots, m\}$ is alignable, then M is also alignable.

Proof. Let (R_i, f_i, g_i) be an aligned tree of M_i . Hence the embeddings $f_i : S_i \rightarrow R_i$ and $g_i : T_i \rightarrow R_i$ satisfy $f_i(s) = g_i(t)$ for all $(s, t) \in M_i$ for $i \in \{1, \dots, m\}$. Now we let

$$V(R) = \{r\} \cup \bigcup_{i=1}^m V(R_i) \cup \bigcup_{i=m+1}^n V(T_i),$$

where r is a new node not in S nor T . We define the relation \leq_R so that, for $x, y \in R$, $x \leq_R y$ if and only one of the following holds.

1. $1 \leq i \leq m$, $x, y \in R_i$ and $x \leq_{R_i} y$;
2. $m < i \leq n$, $x, y \in T_i$ and $x \leq_{T_i} y$;
3. $y = r$.

Note that $R = (V(R), \leq_R)$ is a tree.

Let $\alpha_i : V(R_i) \rightarrow V(R)$ for $i \in \{1, \dots, m\}$ and $\beta_j : V(T_j) \rightarrow V(R)$ for $j \in \{m+1, \dots, n\}$ be the natural inclusions. Thus we define $f : V(S) \rightarrow V(R)$ and $g : V(T) \rightarrow V(R)$ as follows:

$$f(x) = \begin{cases} \alpha_i(f_i(x)) & \text{if } x \in S_i \\ r & \text{if } x = \text{root}(S), \end{cases}$$

$$g(x) = \begin{cases} \alpha_i(g_i(x)) & \text{if } x \in T_i \text{ for } i \in \{1, \dots, m\} \\ \beta_i(x) & \text{if } x \in T_i \text{ for } i \in \{m+1, \dots, n\} \\ r & \text{if } x = \text{root}(S). \end{cases}$$

It is easy to see that f and g are embeddings. Since $f(s) = \alpha_i(f_i(s)) = \alpha_i(g_i(t)) = g(t)$ for $(s, t) \in M_i$, the lemma is shown. \blacksquare

Now we are ready to prove the following important theorem.

Theorem 4.19 For any tree mapping M , the following two properties are equivalent, i.e $ALN = CST^\sharp$.

1. M is alignable (in Definition 4.11).
2. M is less-constrained (in Definition 4.8).

Proof. (1 \Rightarrow 2): Let (R, f, g) be an alignment of M . Then $f : S \rightarrow R$ and $g : T \rightarrow R$ are embeddings such that $f(s) = g(t)$ for all $(s, t) \in M$. Further \bar{g} denote the contraction such that $\bar{g} \circ g$ is the identity map of T (Theorem 3.37).

Suppose that (s_1, t_1) , (s_2, t_2) , and (s_3, t_3) are any three elements of M such that $s_1 \sim s_2 < s_1 \sim s_3$. We have $f(s_1) \sim f(s_2) < f(s_1) \sim f(s_3)$ by Corollary 3.20, and therefore $f(s_2) \sim f(s_3) = f(s_1) \sim f(s_3)$. Also, we have $\bar{g}(f(s_2)) \sim \bar{g}(f(s_3)) = \bar{g}(f(s_2)) \sim f(s_3) = \bar{g}(f(s_1)) \sim \bar{g}(f(s_3))$ by Proposition 3.28.

Since $\bar{g}(f(s_1)) = t_1$, $\bar{g}(f(s_2)) = t_2$ and $\bar{g}(f(s_3)) = t_3$ hold by Lemma 4.13, we conclude that $t_2 \sim t_3 = t_1 \sim t_3$. Derivation of $s_2 \sim s_3 = s_1 \sim s_3$ from $t_1 \sim t_2 < t_1 \sim t_3$ is shown in the same way.

(2 \Rightarrow 1): We prove this assertion by induction on the size of the tree mapping M . In the case of $|M| = 1$, this assertion directly follows Proposition 4.15.

Let $|M| \geq 2$ for the induction step. Let M be the set of node pairs $\{(s_1, t_1), \dots, (s_n, t_n)\}$,

$$X = \{s_1, \dots, s_n\} \subseteq V(S), \text{ and } Y = \{t_1, \dots, t_n\} \subseteq V(T).$$

It suffices to prove the assertion of the theorem under the hypothesis that $\text{lca}(X) = \text{root}(S)$ and $\text{lca}(Y) = \text{root}(T)$. In fact, for the embeddings

$$\alpha = \mathbf{E}_{V(S(\text{lca}(X)))} : S(\text{lca}(X)) \rightarrow S, \text{ and}$$

$$\beta = \mathbf{E}_{V(T(\text{lca}(Y)))} : T(\text{lca}(Y)) \rightarrow T,$$

Lemma 4.16 asserts that, if $M' = \{(\alpha^{-1}(s), \beta^{-1}(t)) \mid (s, t) \in M\}$ is alignable, then M is alignable.

Also, we may assume that M does not contain $(\text{root}(S), \text{root}(T))$ since if M contains it, we have only to eliminate it by Lemma 4.17.

We now choose $X_k = \{s_1, \dots, s_k\}$, by reordering s_i 's if necessary, so that

- $k \geq 1$,
- $\text{lca}(X_k)$ is not the root of S ,
- for any $x \in X \setminus X_k$, $\text{lca}(X_k \cup \{x\}) = \text{root}(S)$.

Note that $k < n$. Let us denote by Y_k the set of nodes $\{t_1, \dots, t_k\}$ corresponding to X_k .

Claim 1 For any $i \leq k$ and $j > k$, the node $s_i \sim s_j$ is the root of S .

Proof. The two nodes $s_i \sim s_j$ and $\text{lca}(S_k)$ are comparable since $s_i \in X_k$. Now assume that $s_i \sim s_j \leq \text{lca}(X_k)$. It follows that $\text{lca}(S_k \cup \{s_j\}) = \text{lca}(X_k)$. This contradicts the definition of X_k . Hence $\text{lca}(X_k) < s_i \sim s_j$, and in particular $s_i \sim s_j = \text{lca}(X_k \cup \{s_j\})$. This implies that $s_i \sim s_j$ is the root of S . ■

Let $A = \{x \in \text{ch}(\text{root}(S)) \mid \exists i[1 \leq i \leq k \wedge s_i \leq x]\}$, and $B = \{x \in \text{ch}(\text{root}(S)) \mid \exists j[k < j \leq n \wedge s_j \leq x]\}$. We have $A \cap B = \emptyset$ since if $x \in A \cap B$, we have $s_i \sim s_j \leq x$ for $1 \leq i \leq k$ and $k < j \leq n$, as is contradictory to Claim 1.

Thus, by inserting nodes as children of $\text{root}(S)$ if necessary, we may assume the following properties (Lemma 4.16 asserts that, if M is alignable after insertion of nodes, it is alignable without the insertion):

- the children of $\text{root}(S)$ are only two nodes v and w ,
- $\text{lca}(S_k) \leq v$,
- $\text{lca}(X \setminus S_k) \leq w$.

Now, to apply the similar proof to Y_k , we claim the following.

Claim 2 For any $i \leq k$ and $j > k$, the node $t_i \sim t_j$ is the root of T .

Proof. We start with showing that, for any $i' \leq k$ and $j' > k$, $t_i \sim t_j = t_{i'} \sim t_{j'}$. By Claim 1, we now have $s_i \sim s_{i'} < s_i \sim s_j$. Hence, since M is less-constrained (Definition 4.8), $t_{i'} \sim t_j = t_i \sim t_j$ holds. In the same way, we have $s_j \sim s_{j'} \leq b < s_{i'} \sim s_j$ and therefore $t_{i'} \sim t_j = t_{i'} \sim t_{j'}$. Hence, we conclude $t_i \sim t_j = t_{i'} \sim t_{j'}$. Hence we have $t_i \sim t_j = t_{i'} \sim t_{j'}$. Next, we show the assertion of the claim. Since $t_i \sim t_j = t_{i'} \sim t_{j'}$ for all $i' \leq k$ and $j' > k$, we have $\text{lca}(Y) \leq t_i \sim t_j$. Since $\text{lca}(Y)$ is the root of T , the node $t_i \sim t_j$ is also the root of T . ■

Therefore, in the same way as the case of S , by inserting nodes as children of $\text{root}(T)$ if necessary, we may assume the following properties:

- the children of $\text{root}(T)$ are only two nodes v' and w' ,
- $\text{lca}(Y_k) \leq v'$,
- $\text{lca}(Y \setminus Y_k) \leq w'$.

By the induction hypothesis, $M_k = \{(s_1, t_1), \dots, (s_k, t_k)\}$ is an alignable mapping from $S(v)$ to $T(v')$, and $M \setminus M_k$ is an alignable mapping from $S(w)$ to $T(w')$. Then, by Lemma 4.18, M is an alignable mapping from S to T . ■

For ordered trees, an algorithm for computing a less-constrained edit distance was presented by Lu *et al.* [LST01]. As in The time complexity of the algorithm is, for two trees T_1 and T_2 ,

$$O(|T_1| \cdot |T_2| \cdot \deg(T_1)^3 \cdot \deg(T_2)^3 \cdot (\deg(T_1) + \deg(T_2)))$$

. By Theorem 4.19, we can immediately improve this algorithm because there is a more efficient algorithm for computing an alignment of trees by [JWZ95]. The time complexity is

$$O(|T_1| \cdot |T_2| \cdot (\deg(T_1) + \deg(T_2))^2)$$

Recall that Jiang *et al.* showed that the alignment problem for two unordered trees is MAX SNP-hard [JWZ95]. Furthermore, we obtain a more negative result for the alignment of trees because Lu *et al.* showed that the less-constrained distance problem for unordered trees has no polynomial-time absolute approximation algorithm [LST01], i.e. the solution is not within an additive constant of the optimum, unless $P = NP$. Then we immediately have the following corollary.

Corollary 4.20 The alignment problem for two unordered trees has no polynomial-time absolute approximation algorithm, unless $P = NP$.

4.6.3 Property of Alignable Mappings

In order to verify that a given tree mapping M is alignable, we do not need to verify that the requisite condition for the alignable mapping holds for all the elements of M . In fact, it suffices to verify the same condition only for the *leaves* of M defined as follows.

Definition 4.21 (Leaves in Tree Mapping) For a tree mapping M , the set of leaves $\mathcal{L}(M)$ is defined as

$$\mathcal{L}(M) = \{(s, t) \in M \mid \forall (x, y) \in M [x \leq s \implies x = s]\}.$$

Proposition 4.22 Let M be a Tai mapping. Then, the following are equivalent.

1. M is alignable.
2. $\mathcal{L}(M)$ is alignable.

Proof. (1 \Rightarrow 2): It is obvious.

(2 \Rightarrow 1): For $(s_1, t_1), (s_2, t_2), (s_3, t_3) \in M$, we assume that $s_1 \sim s_2 < s_1 \sim s_3$. Further, let $(s'_i, t'_i) \in \mathcal{L}(M)$ satisfy $s'_i \leq s_i$ and $t'_i \leq t_i$ for each $i \in \{1, 2, 3\}$. We consider all the cases as follows.

- (i) If s_i and s_j are not comparable with each other for any $i, j \in \{1, 2, 3\}$ such that $i \neq j$, we have $s_i \sim s_j = s'_i \sim s'_j$ and $t_i \sim t_j = t'_i \sim t'_j$ by Proposition 3.1. This immediately implies the assertion.
- (ii) If $s_1 \leq s_3$ and $s_2 \leq s_3$ hold, then $t_1 \sim t_3 = t_2 \sim t_3 = t_3$ holds.
- (iii) If $s_1 \leq s_2$ holds, then $s_1 \sim s_2 = s_1 \sim s_3$ implies that s_2 is not an ancestor of s_3 . Hence, we have $t_1 \sim t_2 = t_2 < t_2 \sim t_3$.

Therefore, we have the assertion. ■

This proposition implies a close relation to the analysis of phylogenetic trees although it goes beyond the scope of this thesis.

4.7 Semi-Accordant Mappings: $\text{Acc}^\sharp = \text{Cst} = \text{SP}^b = \text{SR}$

We have already shown that the following three mappings are equivalent: strongly structure-preserving, constrained, and structure-respecting mappings. Therefore, we unify these three classes of tree mappings, and coin a new term standing for these classes instead of the conventional terms.

As shown in this section title, we refer to the class of these three mappings as *semi-accordant mapping*, and denote the class by Acc^\sharp . We employ the definition of constrained mapping as the primary definition of semi-accordant mapping.

Definition 4.23 (Semi-Accordant Mapping) A Tai mapping M is *semi-accordant* if the following condition holds:

$$\forall (s_1, t_1), (s_2, t_2), (s_3, t_3) \in M [s_3 < s_1 \sim s_2 \iff t_3 < t_1 \sim t_2].$$

In order to verify that a given tree mapping M is semi-accordant, we do not need to verify that the requisite condition for the semi-accordant mapping holds for all the elements of M . As in case of the alignable mapping, it suffices to verify the same condition only for the *leaves* of M defined in Definition 4.21.

Proposition 4.24 Let M be a Tai mapping. Then, the following are equivalent.

1. M is semi-accordant.
2. $\mathcal{L}(M)$ is semi-accordant.

Proof. (1 \Rightarrow 2): It is obvious.

(2 \Rightarrow 1): For $(s_1, t_1), (s_2, t_2), (s_3, t_3) \in M$, we assume that $s_1 \sim s_2 = s_1 \sim s_3$. Further, let $(s'_i, t'_i) \in \mathcal{L}(M)$ satisfy $s'_i \leq s_i$ and $t'_i \leq t_i$ for each $i \in \{1, 2, 3\}$. Since s_i and s_j are not comparable with each other for any $i, j \in \{1, 2, 3\}$ such that $i \neq j$, we have $s_i \sim s_j = s'_i \sim s'_j$ and $t_i \sim t_j = t'_i \sim t'_j$ by Proposition 3.1. This immediately implies the assertion. \blacksquare

4.8 Accordant and Lu Mappings: $\text{Acc} \supsetneq \text{Acc}^* = \text{Lu}$

Here, we introduce an important class of tree mapping by restricting accordant mapping. This class plays an important role in Part II in this thesis.

Definition 4.25 (Accordant Mapping) For a Tai mapping M , M is *accordant* if and only if the following condition is satisfied:

$$\forall (s_1, t_1), (s_2, t_2), (s_3, t_3) \in M [s_1 \sim s_2 = s_1 \sim s_3 \iff t_1 \sim t_2 = t_1 \sim t_3].$$

Example 4.26 Figure 4.3 shows an example of accordant mappings. Figure 4.3(a) and (b) are accordant while Figure 4.3(c) is semi-accordant but not accordant. \blacksquare

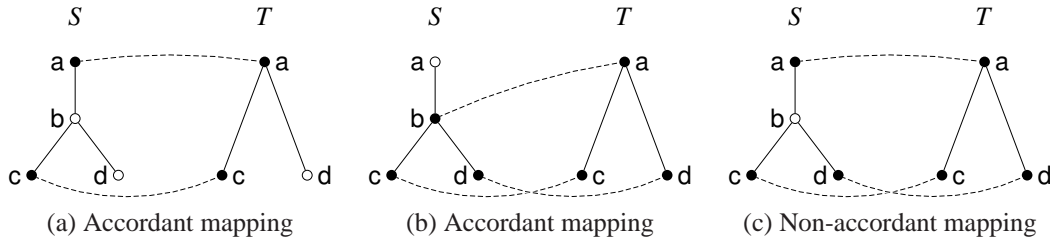


Figure 4.3. Accordant mapping

The accordant mapping has the following equivalent form of definitions.

Proposition 4.27 For a Tai mapping M , the following four conditions are all equivalent.

1. $\forall (s_1, t_1), (s_2, t_2), (s_3, t_3) \in M [s_1 \sim s_2 = s_1 \sim s_3 \iff t_1 \sim t_2 = t_1 \sim t_3].$
2. $\forall (s_1, t_1), (s_2, t_2), (s_3, t_3) \in M [s_1 \sim s_2 < s_1 \sim s_3 \iff t_1 \sim t_2 < t_1 \sim t_3].$
3. $\forall (s_1, t_1), (s_2, t_2), (s_3, t_3), (s_4, t_4) \in M [s_1 \sim s_2 = s_3 \sim s_4 \iff t_1 \sim t_2 = t_3 \sim t_4].$
4. $\forall (s_1, t_1), (s_2, t_2), (s_3, t_3), (s_4, t_4) \in M [s_1 \sim s_2 < s_3 \sim s_4 \iff t_1 \sim t_2 < t_3 \sim t_4].$

Proof. (1 \Rightarrow 2): If $s_1 \sim s_2 < s_1 \sim s_3$, then $s_2 \sim s_3 = s_1 \sim s_3$. By condition (1), we have $t_2 \sim t_3 = t_1 \sim t_3$ and therefore $t_1 \sim t_2 \leq t_2 \sim t_3$. Since $s_1 \sim s_2 = s_1 \sim s_3$ follows $t_1 \sim t_2 = t_1 \sim t_3$, we conclude $t_1 \sim t_2 < t_1 \sim t_3$.

(2 \Rightarrow 1): We claim that $s_1 \sim s_2 \neq s_1 \sim s_3$ if $t_1 \sim t_2 \neq t_1 \sim t_3$. Since $t_1 \sim t_2$ and $t_1 \sim t_3$ are comparable, we may assume that $t_1 \sim t_2 < t_1 \sim t_3$. Hence, we have $s_1 \sim s_2 < s_1 \sim s_3$ by condition (2).

(1 \wedge 2 \Rightarrow 3 \wedge 4): By Proposition 3.1, we may assume $s_1 \sim s_3 = s_3 \sim s_4$ and therefore $t_1 \sim t_3 = t_3 \sim t_4$ by condition (1). $t_1 \sim t_2 = t_3 \sim t_4$ follows $s_1 \sim s_2 = s_1 \sim s_3$ by condition (1), and $t_1 \sim t_2 < t_3 \sim t_4$ does $s_1 \sim s_2 < s_1 \sim s_3$ by condition (2).

(3 \Rightarrow 1 and 4 \Rightarrow 2): The conditions (1) and (2) are respectively obtained by letting $s_1 = s_4$ in the conditions (3) and (4). \blacksquare

If Eq.(2.7a) of the recurrences in Section 2.8.3 is rewritten as follows, the algorithm for computing an optimal accordant mapping is immediately obtained.

$$D_T(v_1(F_1), v_2(F_2)) = \min \begin{cases} D_T(\emptyset, v_2(F_2)) + \min_{T \in F_2} \{D_T(v_1(F_1), T) - D_T(\emptyset, T)\} \\ D_T(v_1(F_1), \emptyset) + \min_{T \in F_1} \{D_T(T, v_2(F_2)) - D_T(T, \emptyset)\} \\ D(F_1, F_2) + d(v_1, v_2) \\ \boxed{D(F_1, F_2) + d(v_1, \varepsilon) + d(\varepsilon, v_2)} \end{cases}$$

Without proof, we show some important properties of accordant mappings.

Proposition 4.28 For an accordant mapping M , the following hold.

1. M is monotonic, symmetric, and transitive.
2. If d is metric, accordant distance is a metric.
3. If M is accordant, M is semi-accordant, and not vice versa; i.e $\text{Acc}^\# \supseteq \text{Acc}$.

4.8.1 Closure of Tree Mappings

). The accordant mapping is characterized by the existence of the *closure* with respect to the binary operation $x \sim y$. Hence, there exists a tree mapping M^* such that $M \subseteq M^*$ and $(s_1 \sim s_2, t_1 \sim t_2) \in M^*$ for any $(s_1, t_1), (s_2, t_2) \in M^*$ if and only if M^* is accordant.

Definition 4.29 (Closure of Set of Nodes) For a given subset $U \subseteq V(T)$, we can define the *closure* U^* of U as the minimum set satisfying the following:

1. $V(T) \supseteq U^* \supseteq U$.
2. For any $x, y \in U^*$, $x \sim y \in U^*$.

From this definition, the closure U^* of U is defined as follows:

$$U^* = V(T) \cup \{x \sim y \mid x, y \in U\}.$$

We extend the notion of closure to tree mappings.

Definition 4.30 (Closure of Tree Mapping) For two trees S and T , let M be a tree mapping between S and T . The closure of M is defined as the minimum tree mapping satisfying the following.

1. $V(S) \times V(T) \supseteq M^* \supseteq M$.
2. $(s_1 \sim s_2, t_1 \sim t_2) \in M^*$ holds for any $(s_1, t_1), (s_2, t_2) \in M^*$.

If the closure M^* exists for a tree mapping M , the following holds:

$$M^* = M \cup \{(s_1 \sim s_2, t_1 \sim t_2) \mid (s_1, t_1), (s_2, t_2) \in M\}.$$

We refer to the class of the closure of \mathcal{C} -mapping as \mathcal{C}^* . Note that every tree mapping does not necessarily have its closure. In fact, a tree mapping has its closure if and only if it is accordant.

Example 4.31 In **Figure 4.4**, we show an accordant mapping (left), and its closure (right). It is easy to see that the closure is also an accordant mapping. In **Figure 4.5**(left), we show a Tai mapping. **Figure 4.5**(right) shows that the resulting mapping calculated along the definition of closure in **Definition 4.30**. It is obvious that this mapping is not a tree mapping. Then there is no closure for the Tai mapping in **Figure 4.5**(left). ■

Proposition 4.32 For a tree mapping M , the following are equivalent.

1. M is accordant.
2. $M^* = M \cup \{(s_1 \sim s_2, t_1 \sim t_2) \mid (s_1, t_1), (s_2, t_2) \in M\}$ is a tree mapping.
3. M^* is accordant

Proof. The equivalence between (1) and (2) is obvious by (3) and (4) in **Proposition 4.27**. The condition (3) apparently implies (2).

(3) \Rightarrow 2): $M^* \cup \{(s_1 \sim s_2, t_1 \sim t_2) \mid (s_1, s_2), (t_1, t_2) \in M^*\} = M^*$ holds. In particular, $M^* \cup \{(s_1 \sim s_2, t_1 \sim t_2) \mid (s_1, s_2), (t_1, t_2) \in M^*\}$ is a tree mapping, and hence M^* is accordant. ■

We have immediately the following proposition from the definition of closure of tree mappings.

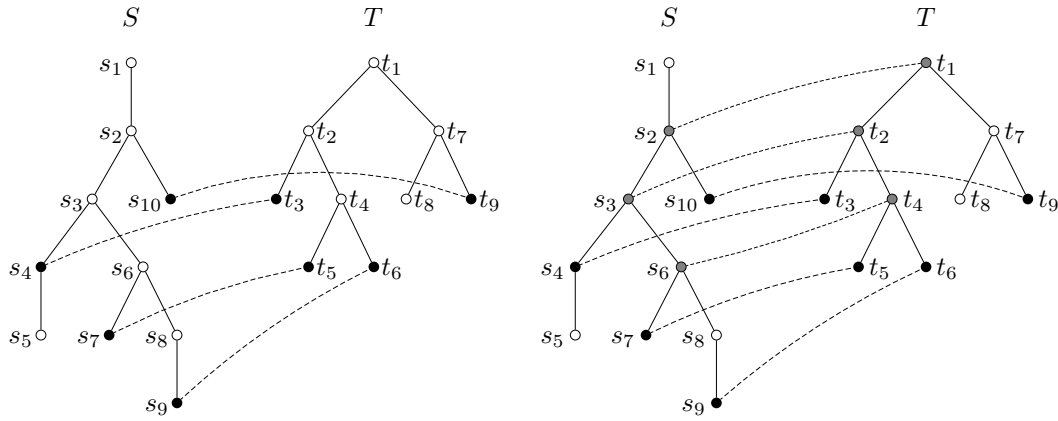


Figure 4.4. Closure of a tree mapping

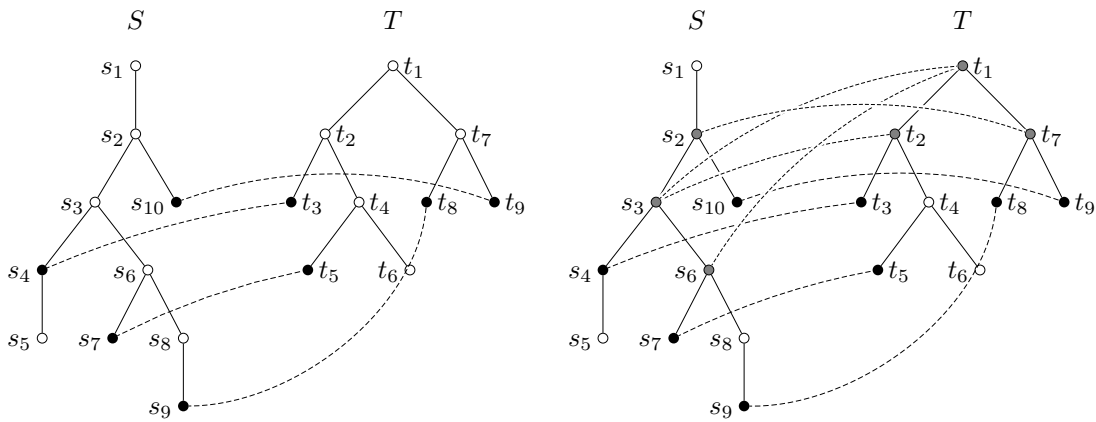


Figure 4.5. A Tai mapping with no closure

Proposition 4.33 (Tree Mapping for Lu distance) A tree mapping in Lu distance is the closure of an accordant mapping, i.e.

$$\text{Acc} \supseteq \text{Acc}^* = \text{Lu}.$$

If d is a metric, the following holds:

$$D^{\text{Acc}}(T_1, T_2) = D^{\text{Acc}^*}(T_1, T_2).$$

4.9 Summary

We have addressed the problems raised in Chapter 2, and untied confusion in the definitions and relationships of tree edit distance measures. The class hierarchy of tree mappings established this chapter is shown in **Figure 4.6**. In **Table 4.1**, we summarize the properties in each class of tree mappings. For tree mapping classes of Tai, alignable, semi-accordant, and accordant, each class is clearly discriminated by a simple tree mapping depicted in Table 4.2.

Table 4.1. Properties of tree mapping classes

Class	Equivalent Class (ID)	Reference	Metricity	Transitivity	Symmetry	Monotonicity
Tai (TAI)	Tai (TAI)	[Tai79]	✓	✓	✓	✓
Alignable (ALN)	Alignment (ALN) Less-Constrained (Cst [‡])	[JWZ95] [LST01]			✓	✓
Structure-Preserving (SP)	Structure-Preserving (SP)	[TT88]		✓		✓
Semi-Accordant (Acc [‡])	Strongly Structure-Preserving (SP [‡]) Constrained (CST) Structure-Respecting (SR)	[Tan84] [Zha95, Zha96] [Ric97]	✓	✓	✓	✓
Accordant (Acc)		This thesis	✓	✓	✓	✓
Closure of Accordant (Acc*)	Lu (LU) Glycan Matching	[Lu79] [AYO ⁺ 03]	✓	✓	✓	
Top-Down (Top)	Top-Down (Top) LCST (LCST)	[Sel77] [Aku92]	✓	✓	✓	✓
Bottom-Up (Bot)	bottom-up (Bot)	[Val98]	✓	✓	✓	✓
Inclusion (INCLUSION)	Inclusion (INCLUSION)	[KM95]		✓	left-total	
Constrained Inclusion (INCLUSION [‡])	Constrained Inclusion (INCLUSION [‡])	[Val05]		✓	left-total	
Isomorphism (ISOMORPHISM)	Isomorphism (ISOMORPHISM)		✓	✓	✓	✓

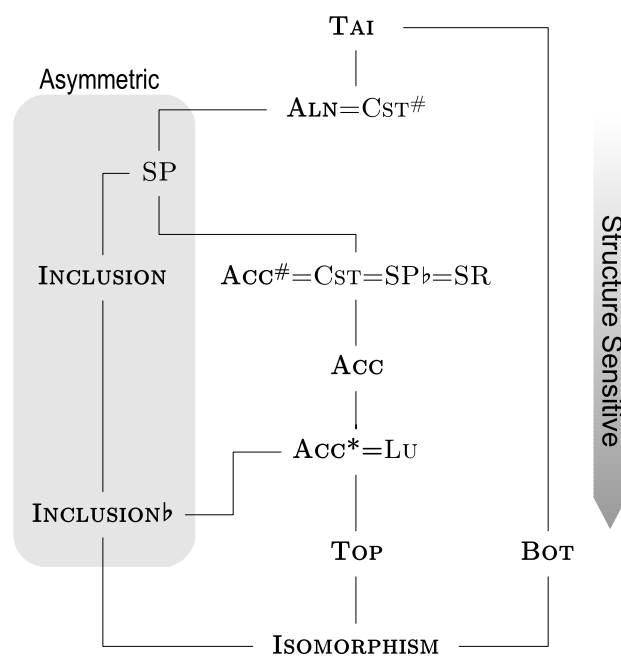
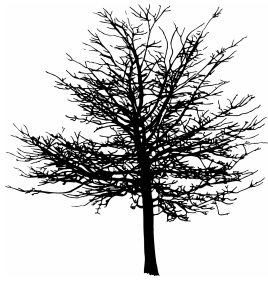


Figure 4.6. Class hierarchy of tree mappings

Table 4.2. *Characteristic of tree mapping classes*

S					
T					
	Tai Alignable Semi-Acc. Accordant	Tai Alignable	Tai		
	Tai Alignable	Tai Alignable Semi-Acc. Accordant	Tai Alignable		
	Tai	Tai Alignable	Tai Alignable Semi-Acc. Accordant		
				Tai Alignable Semi-Acc. Accordant	Tai Alignable Semi-Acc.
				Tai Alignable Semi-Acc.	Tai Alignable Semi-Acc. Accordant

Semi-Acc. stands for Semi-Accordant,
 $M = \{(s_1, t_1), (s_2, t_2), (s_3, t_3)\}$.



Part II

Learning in Trees

“I like to climb trees. I can see everything.”

— Orson Scott Card, *Speaker for the Dead*



Chapter 5

Kernel-based Learning for Trees

Data encountered in the real world are often not represented as vectors of numbers, but as structured data. Analysis of *structured data* such as sequences, trees, and graphs is attracting considerable attention. Hausler [Hau99] introduced the *convolution kernel*, a general framework for designing kernel functions for discrete data structures including structured data. The basic idea of convolution kernel is to decompose a data object into its parts, and to define a kernel function in terms of the parts. Many convolution kernels specialized for various discrete data structures have been proposed.

In this chapter, we focus on the kernel method for *rooted ordered labeled trees*. A rooted ordered labeled tree is a fairly general data structure that models a wide variety of structured data including parse trees of natural language texts, semi-structured data such as HTML/XML, and biological data such as RNA secondary structures and glycans. We focus on glycans in the next chapter. Throughout this chapter, we refer to rooted ordered labeled trees simply as *trees* unless otherwise stated.

5.1 Support Vector Machines

Support Vector Machines (SVMs) are a class of supervised learning algorithms first introduced by Vapnik [Vap95]. SVMs have shown an outstanding generalization performance in a variety of practical problems and have a strong theoretical basis in statistical learning theory.

SVMs is primarily a two-class discriminative classifier. For a given set of training data, each of which is labeled with positive and negative, SVMs learn a linear decision boundary to discriminate between the positive and negative classes of the predetermined training data. To determine whether an arbitrary datum is positive or negative, which is not necessarily a training datum, the decision boundary has only to be applied to each element of data.

To be more precise, the two-class classification is defined as follows. Let $X \subset \mathbb{R}^n$ denote the input space and Y denote the output space $\{-1, +1\}$, where the values -1 and $+1$ indicate the labels of negative and positive respectively.

A *training datum* is a pair in $X \times Y$ and a training set is a finite set of training data, denoted by

$$D = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)\} \subsetneq X \times Y.$$

The \mathbf{x}_i is referred to as an example, and the y_i as a *class label*. The purpose of the learning procedure in SVMs is to find a hypothesis function $f : X \rightarrow \mathbb{R} \setminus \{0\}$, which is characterized by a pair of parameters, *weight vector* $\mathbf{w} \in \mathbb{R}^n$ and *threshold* $b \in \mathbb{R}$. In fact, the hypothesis function is represented by the following linear function:

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$$

A datum \mathbf{x} is classified as positive or negative if $f(\mathbf{x}) < 0$ or $f(\mathbf{x}) > 0$ respectively. In other words, the learning procedure outputs a *decision function* $f_d : X \rightarrow Y$, i.e.

$$f_d(\mathbf{x}) = \text{sgn}(\mathbf{w} \cdot \mathbf{x} + b),$$

so that $y_i = f_d(\mathbf{x}_i)$ approximates the probabilistic relation between inputs and outputs.

5.2 Kernel Methods

The *kernel method*, a method of machine learning, provides a diversity of applications with a generic and flexible framework to solve various problems including the classification problem, and is being extensively studied (cf. [STC04]). The advantages of the kernel method, also known as the *kernel trick*, have resulted from the usage of *kernel functions* instead of usage of explicit and fixed vector representations of data.

We take the classification problem for instance to clarify the feature of the kernel method. The classification problem is a problem to find a decision function

$$f_d : \mathcal{X} \rightarrow \{-1, 1\}$$

so that $f_d(x_i) = y_i$ holds for a given set of training data $\{(x_1, y_1), \dots, (x_N, y_N)\} \subset \mathcal{X} \times \{1, -1\}$ for a proper subset $\{x_1, \dots, x_N\}$ of a data space \mathcal{X} , i.e. $\{x_1, \dots, x_N\} \subsetneq \mathcal{X}$. As seen in the previous section, SVMs solve the classification problem, if the set of training data $\{(x_i, y_i)\}$ is *linearly separable*, i.e. under a given vector representation of an object space X by $\phi : \mathcal{X} \rightarrow \mathbb{R}^n$, there exist $\mathbf{w} \in \mathbb{R}^n$ and $b \in \mathbb{R}$ such that:

$$\forall i [y_i = \text{sgn}(\mathbf{w} \cdot \phi(x_i) + b)]. \quad (5.1)$$

Eventually, given ϕ and the training data, SVMs find \mathbf{w} and b that make Eq.(5.1) hold. When $\phi_i(x)$ denotes the i -th component of $\phi(x)$, i.e. $\phi(x) = (\phi_1(x), \dots, \phi_n(x))$, each $\phi_i(x)$ is called a *feature* of x , and \mathbb{R}^n is a *feature space*.

A problem in practicing SVMs is that it is an elaborate and time-consuming job to *discover* an appropriate $\phi : \mathcal{X} \rightarrow \mathbb{R}^n$ so that the image of the positive training data (i.e. $\{\phi(x_i) \mid y_i = 1\}$) and that of the negative ones (i.e. $\{\phi(x_i) \mid y_i = -1\}$) are split by a hyperplane.

Apart from the classification problem and SVMs, the same applies to many conventional methods of machine learning. In fact, they solve problems provided with a feature space and a vector representation that fulfill certain *good* conditions. In other words, to take advantage of conventional learning machines, it is required to determine a *good* vector representation ϕ to a feature space, which is an elaborate and time-consuming job, too.

In contrast, the kernel method does not require a vector representation ϕ , but does a kernel instead, which is a symmetric and positive semidefinite *pairing* $\mathbf{K} : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$.

Symmetric: $\mathbf{K}(x, y) = \mathbf{K}(y, x)$;

Positive Semidefinite: an arbitrary *Gram matrix*

$$[\mathbf{K}(x_i, x_j)]_{i,j=1,\dots,n} = \begin{bmatrix} \mathbf{K}(x_1, x_1) & \dots & \mathbf{K}(x_1, x_n) \\ \vdots & \ddots & \vdots \\ \mathbf{K}(x_n, x_1) & \dots & \mathbf{K}(x_n, x_n) \end{bmatrix}$$

has only non-negative eigenvalues.

In learning machines such as SVMs, we need to provide an appropriate vector representation ϕ such that the provided kernel \mathbf{K} is identical to the natural inner product in the associated feature space, that is, $\mathbf{K}(x, y) = \langle \phi(x), \phi(y) \rangle$ holds. The kernel method encapsulates vector representations of data and feature spaces, and does not necessarily require *explicit* vector representations. We have only to design a *similarity function* $\mathbf{K}(x, y)$ between two objects x and y , and verify the positive semidefiniteness of the function.

5.3 Haussler's Convolution Kernels

Haussler [Hau99] proposed *convolution kernels*, a general framework for handling discrete data structures by kernel methods. The basic idea of the Haussler's convolution kernel method is

1. decomposing objects into their *subparts* and then,
2. calculating the convolution kernel using a known underlying kernel defined for the subparts (a subpart kernel).

Let \mathcal{X} and \mathcal{X}' denote nonempty data spaces. The following theorem gives the special form of the Haussler's R -convolution kernel [Hau99, Theorem 1] for the case of $D = 1$.

Theorem 5.1 (Convolution Kernel with $D = 1$ [Hau99]) Let $\mathbf{K}' : \mathcal{X}' \times \mathcal{X}' \rightarrow \mathbb{R}$ be a kernel. Given a binary relation $R \subset \mathcal{X}' \times \mathcal{X}$, the function $\mathbf{K} : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ defined by the following is also a kernel.

$$\mathbf{K}(x, y) = \sum_{(x', x) \in R} \sum_{(y', y) \in R} \mathbf{K}'(x', y').$$

Intuitively, let $\mathcal{S}(x)$ be the set including all parts of the structure x such that $\mathcal{S} : \mathcal{X} \rightarrow 2^{\mathcal{X}'}$, the convolution kernel is defined as follows:

$$\mathbf{K}(x, y) = \sum_{x' \in \mathcal{S}(x)} \sum_{y' \in \mathcal{S}(y)} \mathbf{K}'(x', y').$$

Haussler's theorem [Hau99, Theorem 1], which defines the general form of the R -convolution kernel, is obtained as a corollary to Theorem 5.1.

Corollary 5.2 (Haussler [Hau99]) Let $\mathbf{K}'_d : \mathcal{X}'_d \times \mathcal{X}'_d \rightarrow \mathbb{R}$ be kernels for $d = 1, \dots, D$. Given a relation $R \subset \mathcal{X}'_1 \times \dots \times \mathcal{X}'_D \times \mathcal{X}$, the function $\mathbf{K} : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ defined in Eq.(5.2) is also a kernel.

$$\mathbf{K}(x, y) = \sum_{(x'_1, \dots, x'_D, x) \in R} \sum_{(y'_1, \dots, y'_D, y) \in R} \prod_{d=1}^D \mathbf{K}'_d(x'_d, y'_d) \quad (5.2)$$

Proof. Define a function $\mathbf{K}' : (\mathcal{X}'_1 \times \dots \times \mathcal{X}'_D) \times (\mathcal{X}'_1 \times \dots \times \mathcal{X}'_D) \rightarrow \mathbb{R}$ as follows.

$$\mathbf{K}'((x'_1, \dots, x'_D), (y'_1, \dots, y'_D)) = \prod_{d=1}^D \mathbf{K}'_d(x'_d, y'_d) \quad (5.3)$$

Since \mathbf{K}' is the tensor product of $\mathbf{K}'_1, \dots, \mathbf{K}'_D$ and the set of kernels is closed under the operation of tensor product, \mathbf{K}' is a kernel over $\mathcal{X}' = \mathcal{X}'_1 \times \dots \times \mathcal{X}'_D$. Therefore, applying Theorem 5.1 to \mathbf{k} defined in Eq.(5.3), we obtain the fact that \mathbf{K} defined by Eq.(5.2) is a kernel. \blacksquare

The set of kernels is closed under direct sum, as well. Therefore, we obtain that \mathbf{K} defined in Eq.(5.4) is a kernel as a corollary to Theorem 5.1.

$$\mathbf{K}(x, y) = \sum_{((x'_1, \dots, x'_D), x) \in R} \sum_{((y'_1, \dots, y'_D), y) \in R} \sum_{d=1}^D \mathbf{K}'_d(x'_d, y'_d) \quad (5.4)$$

The implication of the above discussion is that the variety of the operations that preserve the property of being kernels is an important factor to determine the range of application of the Haussler's convolution kernel. In this regards, Lemma 5.3 and Corollary 5.4 adds an example of the Haussler's convolution kernel, but the meaning is more than it. In fact, Corollary 5.4 showed a way to weight the underlying kernels $\mathbf{K}'(x', y')$ when they are accumulated into a convolution kernel.

Haussler's theorem is available even if the underlying kernels $\mathbf{K}'(x', y')$ are weighted according to the structures x' and y' .

Lemma 5.3 Let $\mathbf{K}' : \mathcal{X}' \times \mathcal{X}' \rightarrow \mathbb{R}$ be a kernel. For any function $f : \mathcal{X}' \rightarrow \mathbb{R}$, the function \mathbf{K}'' defined by the following is a kernel.

$$\mathbf{K}''(x, y) = f(x) \cdot f(y) \cdot \mathbf{K}'(x, y) \quad (5.5)$$

Proof. Apparently, $F(x, y) = f(x) \cdot f(y)$ is a kernel. Since the property of being kernels is closed under tensor product and restriction of domains, \mathbf{K}'' is also a kernel. \blacksquare

Corollary 5.4 For a given function $\mathcal{S} : \mathcal{X} \rightarrow 2^{\mathcal{X}'}$, let w_x be a set of functions indexed by $x \in \mathcal{X}$ such that

$$w_x : \mathcal{S}(x) \rightarrow \mathbb{R}.$$

Then, the function \mathbf{K} defined by the following is a kernel.

$$\mathbf{K}(x, y) = \sum_{x' \in \mathcal{S}(x)} \sum_{y' \in \mathcal{S}(y)} w_x(x') \cdot w_y(y') \cdot \mathbf{K}'(x', y')$$

Proof. By letting \mathcal{X}^* denote $\mathcal{X}' \times \mathcal{X}$, we extend our definitions as follows: $w^* : \mathcal{X}^* \rightarrow \mathbb{R}$, $\mathbf{K}^* : \mathcal{X}^* \times \mathcal{X}^* \rightarrow \mathbb{R}$ and $\mathcal{S}^* : \mathcal{X} \rightarrow 2^{\mathcal{X}^*}$ such that

$$w^*(x', x) = \begin{cases} w(x, x') & \text{if } x' \in \mathcal{S}(x) \\ 0 & \text{if } x' \notin \mathcal{S}(x), \end{cases}$$

$$\mathbf{K}^*((x', x), (y', y)) = \mathbf{K}'(x', y'),$$

$$\mathcal{S}^*(x) = \{(x', x) \mid x' \in \mathcal{S}(x)\}.$$

Since the following Equation holds, the assertion immediately follows from Lemma 5.3 and Theorem 5.1.

$$\mathbf{K}(x, y) = \sum_{(x', x) \in \mathcal{S}^*(x)} \sum_{(y', y) \in \mathcal{S}^*(y)} w^*(x', x) \cdot w^*(y', y) \cdot \mathbf{K}^*((x', x), (y', y))$$

To normalize the size factor of instances, the following *normalized kernel* is often used:

$$\tilde{\mathbf{K}}(x, y) = \frac{\mathbf{K}(x, y)}{\sqrt{\mathbf{K}(x, x)} \sqrt{\mathbf{K}(y, y)}}.$$

5.3.1 Gap-Weighted String Kernel

The string kernels that Lodhi [LSST⁺02] introduced is an important example of Haussler's convolution kernels. Lodhi took advantage of Corollary 5.4 and introduced string kernels. To evaluate similarity between two strings, this kernel [LSST⁺02] uses the number of the substrings commonly occurring between two strings.

First, let us consider the counting function defined as follows.

$$\mathbf{K}(x, y) = \sum_{s \in \mathcal{S}(x)} \sum_{t \in \mathcal{S}(y)} \delta(s, t) \quad (5.6)$$

The function $\mathbf{K}(x, y)$ defined by Eq.(5.6) returns the number of common substrings of x and y , and is asserted to be a kernel by Theorem 5.1. In Eq.(5.6), $\mathcal{S}(x)$ means that the multiset (or bag) of all subsequences in x , and $\delta(s, t)$ is the Kronecker's delta, defined by

$$\delta(s, t) = \begin{cases} 1 & (s = t) \\ 0 & (s \neq t). \end{cases}$$

The kernel \mathbf{K} in Eq.(5.6), however, involves some drawback due to the property that all the substrings are counted with the same weight. For example, for

$$x = \text{conversion} \quad \text{and} \quad y = \text{convolution},$$

the matching of the first occurrences of "con," that is, the matching between

$$\underline{\text{con}}\text{version} \quad \text{and} \quad \underline{\text{con}}\text{volution},$$

is important, since it indicates that they are both contiguous with the same prefix "con." In contrast, the matching between

$$\underline{\text{con}}\text{version} \quad \text{and} \quad \underline{\text{conv}}\text{olution}$$

is just a coincidence, and should have a much less important impact on the similarity between x and y .

To mitigate the drawback, Lodhi [LSST⁺02] introduces a weighted counting method as defined by Eq.(5.7), which is a kernel by Corollary 5.4.

$$\mathbf{K}(x, y) = \sum_{s \in \mathcal{S}(x)} \sum_{t \in \mathcal{S}(y)} \lambda^{g(s,x)} \cdot \lambda^{g(t,y)} \cdot \delta(s, t) \quad (5.7)$$

In Eq.(5.7), $g(s, x)$ is defined as the length of the subsequence s with gaps in x . The constant λ is called a *decay* factor, and is taken from the interval $(0, 1)$. For $s = \text{con}$ and $x = \text{conversion}$, we have the following weight for each matching:

- $g(s, x) = 3$ for matching conversion, since the sequence “con” spans 3 character-long in x .
- $g(s, x) = 10$ for matching convolution, since the sequence “con” spans 10 character-long in x .

5.3.2 Spectrum Kernel for Strings

Leslie *et al.* introduced a simple and efficient string kernel, the *spectrum kernel* [LEN02], for classifying proteins. The spectrum kernel is based on the simple idea that the more substrings with a fixed length are shared in two strings, the more similar they are. Hence, the notion of the spectrum of a string is often used in approximate string matching [JU91, Ukk93].

Let Σ be a finite alphabet. By Σ^* , we denote the set of all strings over Σ , by Σ^q all strings with length q over Σ . A q -gram is any string with length q in Σ^q .

Recall that $\#x[w]$ is defined as the total number of occurrences of w in x (See Section 2.2.7), i.e.

$$\#x[w] = |\{y \mid x = y \cdot w \cdot z \wedge y, z \in \Sigma^*\}|.$$

The q -gram profile of a string x is the vector $G_q(x) = (\#x[w])_{w \in \Sigma^q}$, indexed by all q -grams w and arranged in lexicographic order of q -grams. It is a feature mapping from an input space Σ^* to $\mathbb{R}^{|\Sigma|^q}$. Then, the q -spectrum kernel of two strings s_1 and s_2 is defined as follows.

$$\begin{aligned} \mathbf{K}_q(s_1, s_2) &= \sum_{w \in \Sigma^q} \#s_1[w] \cdot \#s_2[w] \\ &= \langle G_q(s_1), G_q(s_2) \rangle. \end{aligned}$$

Example 5.5 Let $\Sigma = \{a, b\}$, and consider the 2-spectrum kernel of the strings $s_1 = abaaabaa$ and $s_2 = aababbab$. The 2-gram profiles of s_1 and s_2 are given as follows.

w	aa	ab	ba	bb
$\#s_1[w]$	3	2	2	0
$\#s_2[w]$	1	3	2	1

Hence, $\mathbf{K}_2(s_1, s_2) = \langle (3, 2, 2, 0), (1, 3, 2, 1) \rangle = 13$. ■

The q -spectrum kernel $\mathbf{K}_q(s_1, s_2)$ can be evaluated in time $O(q \cdot (|s_1| + |s_2|))$. Moreover, the spectrum kernel allows SVMs to classify a new string in linear time.

5.4 Tree Kernels

Also, for more complex discrete structures than strings, a variety of convolution kernels are proposed. In this section, we focus on the kernels for trees.

The intuitive idea of tree kernels in common is as follows. For two trees T and τ , let $\#T[\tau]$ denote the total number of occurrences of τ in T . The vector representation of T is denoted by

$$\phi(T) = (\#T[\tau_1], \#T[\tau_2], \#T[\tau_3], \dots),$$

where τ_i ($i \geq 1$) are subtree patterns in T . Then the kernel is given by the inner product of two vectors for two trees T_1 and T_2 :

$$\mathbf{K}(T_1, T_2) = \langle \phi(T_1), \phi(T_2) \rangle = \sum_i \#T_1[\tau_i] \cdot \#T_2[\tau_i].$$

The choice criteria of subtree patterns τ_i lead to a variety of tree kernels.

Note that we omit the weight factor in kernels for simplicity.

5.4.1 Parse Tree Kernel

Collins and Duffy [CD01] presented the *parse tree kernel* as a counting function of common subtrees, which is in the class of convolution kernels [Hau99].

Since a common subtree between two ordered trees T_1 and T_2 uniquely defines a partial mapping between $V(T_1)$ and $V(T_2)$, the parse tree kernel is regarded as a counting function of partial node-to-node mappings between trees. Then the kernel function is basically defined as follows. (Note that the definition is slightly modified from the original so that it can be applied not only to parse trees but also to general rooted labeled ordered trees.)

$$\mathbf{K}(T_1, T_2) = \sum_{v_1 \in V(T_1)} \sum_{v_2 \in V(T_2)} \mathbf{K}^{\text{SC}}(v_1, v_2),$$

where $\mathbf{K}^{\text{SC}}(v_1, v_2)$ is the counting function of the number of the partial mappings f that satisfy the following conditions.

- f is a mapping from a set of nodes $V' \subseteq V(T_1)$ to $V(T_2)$ with $l(f(v_1)) = l(v_2)$.
- Any $v \in V' \setminus \{v_1\}$ satisfies the following conditions:
 1. $v < v_1$ (v is a descendent of v_1),
 2. $\text{par}(v) \in V'$,
 3. $|\text{ch}(\text{par}(v))| = |\text{ch}(f(\text{par}(v)))|$,
 4. If v is the i -th child of $\text{par}(v)$, then $f(v)$ is also the i -th child of $f(\text{par}(v))$.

A partial mapping satisfying the above conditions is referred to as a *subtree-congruent* mapping. The value of $\mathbf{K}^{\text{SC}}(v_1, v_2)$ can be calculated by the following recurrences.

$$\mathbf{K}^{\text{SC}}(v_1(F_1), v_2(F_2)) = \begin{cases} \delta(l(v_1), l(v_2)) \cdot \prod_{i=1}^{|\text{ch}(v_1)|} (1 + \mathbf{K}^{\text{SC}}(T_1^i, T_2^i)) & \text{if } |\text{ch}(v_1)| = |\text{ch}(v_2)|, \\ 0 & \text{otherwise,} \end{cases}$$

where we assume, for $n = |\text{ch}(v_1)|$,

$$\begin{aligned} F_1 &= T_1^1 \bullet \cdots \bullet T_1^n, \\ F_2 &= T_2^1 \bullet \cdots \bullet T_2^n. \end{aligned}$$

The value of the kernel $\mathbf{K}(T_1, T_2)$ can be calculated by dynamic programming in $O(|T_1| \cdot |T_2|)$ time. Note that the subtree-congruent mapping is injective and preserves the hierarchical and the sibling orders between two trees.

5.4.2 Labeled Tree Kernel

The parse tree kernel is too restrictive since it considers only subtrees with the same number of children. Kashima and Koyanagi [KK02] proposed an extended tree kernel by generalizing parse tree kernel [CD01].

Let T_1, T_2 and τ be trees. By $\#T_i[\tau]$ we denote the total number of the occurrences of subtree τ in T_i . Then, the labeled ordered tree kernel [KK02] of T_1 and T_2 is defined as follows:

$$\mathbf{K}(T_1, T_2) = \sum_{\tau \in \mathcal{T}} \#T_1[\tau] \cdot \#T_2[\tau],$$

where \mathcal{T} denotes the universal set of trees. The kernel function is rewritten as

$$\mathbf{K}(T_1, T_2) = \sum_{v_1 \in V(T_1)} \sum_{v_2 \in V(T_2)} \#M^{\text{Top}}(T_1(v_1), T_2(v_2)),$$

where $\#M^{\text{Top}}(T_1(v_1), T_2(v_2))$ is the number of common subtrees rooted at both v_1 and v_2 . Specifically, $\#M^{\text{Top}}(T_1(v_1), T_2(v_2))$ is the counting function of the number of the *top-down mappings* M between $T_1(v_1)$ and $T_2(v_2)$. We have the following recurrences for efficiently computing $\#M^{\text{Top}}(T_1(v_1), T_2(v_2))$ by dynamic programming. In the recurrences, let $\mathbf{K}^T(T_1(v_1), T_2(v_2))$ be $\#M^{\text{Top}}(T_1(v_1), T_2(v_2))$.

$$\mathbf{K}(T_1, T_2) = \sum_{v_1 \in V(T_1)} \sum_{v_2 \in V(T_2)} \mathbf{K}_T(T_1(v_1), T_2(v_2)), \tag{5.8a}$$

$$\mathbf{K}_T(v_1(F_1), v_2(F_2)) = \delta(l(v_1), l(v_2)) \cdot (1 + \mathbf{K}_F(F_1, F_2)), \tag{5.8b}$$

$$\mathbf{K}_F(\emptyset, F_2) = \mathbf{K}_F(F_1, \emptyset) = 0, \tag{5.8c}$$

$$\mathbf{K}_F(T_1 \bullet F_1, T_2 \bullet F_2) = \mathbf{K}_F(F_1, T_2 \bullet F_2) + \mathbf{K}_F(T_1 \bullet F_1, F_2) - \mathbf{K}_F(F_1, F_2) + \mathbf{K}_T(T_1, T_2) \cdot (1 + \mathbf{K}_F(F_1, F_2)). \tag{5.8d}$$

We show the algorithm for computing the recurrences in Algorithm 5.1. This algorithm runs in $O(|T_1| \cdot |T_2|)$ time and space.

Example 5.6 Consider two trees T_1 and T_2 in **Figure 5.1**. **Table 5.1** shows the feature vectors of T_1 and T_2 for the labeled tree kernel. In this example, we depict labels of nodes as white circles and black pentagons. We have the following kernel value.

$$\mathbf{K}(T_1, T_2) = \langle (3, 1, 2, 1, 0, 1, 1, 1, 1, 0), (3, 1, 2, 1, 1, 2, 0, 0, 0, 1) \rangle = 16.$$

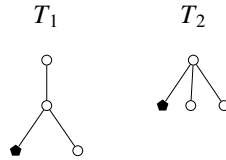


Figure 5.1. Two ordered trees for the labeled tree kernels

Table 5.1. Feature vectors in the labeled tree kernel


	○	●	○	○	○	○	○	○	○	○
T_1	3	1	2	1	0	1	1	1	1	0
T_2	3	1	2	1	1	2	0	0	0	1

Labeled Tree Kernels for Unordered Trees

Kashima *et al.* proved that the problem of computing the labeled tree kernel for unordered trees is #P-complete [KSK06a]. This difficulty can be expected since the problem of counting bipartite matchings is known to be #P-complete [Val79], and tree matching problems for unordered trees inherently include the computation of bipartite matching.

5.4.3 Labeled Tree Kernel with Elastic Structure Matching

Kashima and Koyanagi [KK02] extended their tree kernel by allowing elastic structure matching for a more flexible interpretation of the common patterns, which is referred to as an *elastic tree kernel*.

Now consider two trees T_1 and T_2 in **Figure 5.2**. As in the previous example, we depict labels as white circles and black pentagons. The elastic tree kernel was designed so that the subtree  is also counted as a common structure between T_1 and T_2 .

The *elastic tree kernel* is given by the following recurrences.

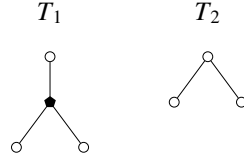


Figure 5.2. Two ordered trees for the elastic tree kernels

Algorithm 5.1 Labeled tree kernels

- $T[i]$: the i -th node in T indexed by postorder numbering ($i \in [1, |T|]$).
- $F[i]$: the i -th tree in F ($i \in [1, |F|]$).
- $\text{ch}(v)$: forest F obtained by removing v from $v(F)$.

```

procedure TREE_KERNEL( $T_1, T_2$ )
  def array  $\mathbf{K}_T[1..|T_1|, 1..|T_2|]$ 
  def array  $\mathbf{K}[1..|T_1| + 1, 1..|T_2| + 1]$  initialized with 0
  for  $i \leftarrow 1$  to  $|T_1|$ 
    for  $j \leftarrow 1$  to  $|T_2|$ 
      if  $l(T_1[i]) \neq l(T_2[j])$  then  $\mathbf{K}_T[i, j] \leftarrow 0$ 
      else  $\mathbf{K}_T[i, j] \leftarrow 1 + \text{FOREST\_KERNEL}(\text{ch}(T_1[i]), \text{ch}(T_2[j]))$ 
       $\mathbf{K}[i + 1, j + 1] \leftarrow \mathbf{K}_T[i, j] + \mathbf{K}[i, j + 1] + \mathbf{K}[i + 1, j] - \mathbf{K}[i, j]$ 
  return  $\mathbf{K}[|T_1| + 1, |T_2| + 1]$ 
end
  
```

```

procedure FOREST_KERNEL( $F_1, F_2$ )
  def array  $\mathbf{K}_F[1..|F_1| + 1, 1..|F_2| + 1]$  initialized with 0
  for  $i \leftarrow 1$  to  $|F_1|$ 
    for  $j \leftarrow 1$  to  $|F_2|$ 
       $\mathbf{K}_F[i + 1, j + 1] \leftarrow \mathbf{K}_F[i + 1, j] + \mathbf{K}_F[i, j + 1]$ 
       $\quad - \mathbf{K}_F[i, j] + \mathbf{K}_T[F_1[i], F_2[j]] * (1 + \mathbf{K}_F[i, j])$ 
  return  $\mathbf{K}_F[|F_1| + 1, |F_2| + 1]$ 
end
  
```

The algorithm for the elastic tree kernel is obtained only by replacing \mathbf{K}_T with \mathbf{K} .

$$\mathbf{K}(T_1, T_2) = \sum_{v_1 \in V(T_1)} \sum_{v_2 \in V(T_2)} \mathbf{K}_T(T_1(v_1), T_2(v_2)), \quad (5.9a)$$

$$\mathbf{K}_T(v_1(F_1), v_2(F_2)) = \delta(l(v_1), l(v_2)) \cdot (1 + \mathbf{K}_F(F_1, F_2)) \quad (5.9b)$$

$$\mathbf{K}_F(\emptyset, F_2) = \mathbf{K}_F(F_1, \emptyset) = 0 \quad (5.9c)$$

$$\begin{aligned} \mathbf{K}_F(T_1 \bullet F_1, T_2 \bullet F_2) = & \mathbf{K}_F(F_1, T_2 \bullet F_2) + \mathbf{K}_F(T_1 \bullet F_1, F_2) - \mathbf{K}_F(F_1, F_2) \\ & + \mathbf{K}(T_1, T_2) \cdot (1 + \mathbf{K}_F(F_1, F_2)). \end{aligned} \quad (5.9d)$$

These recurrences are the same as the recurrences (5.8) except for one letter in Eq.(5.9d). Algorithm 5.1 can be also applied to the computation of the elastic tree kernel with a subtle modification.

In spite of its flexibility, the elastic tree kernel has a certain unintended property. We here discuss the property.



The elastic tree kernel is the counting function of accordant mapping. Although the elastic tree kernel was proposed with the intention of allowing structural ambiguities as much as possible, it was not sufficient. It is easy to prove that the elastic tree kernel [KK02] counts the number of the accordant mappings between two trees. This fact indicates the possibility that more flexible tree kernels can be designed, since we have already seen that there exist more flexible classes of tree mappings. In the next chapter, we construct more flexible tree kernels from the point of view of tree mappings.

 **The elastic tree kernel is not a convolution kernel.** The tree kernel based on convolution is defined as follows:


$$\begin{aligned} \mathbf{K}(T_1, T_2) &= \sum_{\tau_1 \in \mathcal{S}(T_1)} \sum_{\tau_2 \in \mathcal{S}(T_2)} \mathbf{K}'(\tau_1, \tau_2) \\ &= \sum_{(\tau_1, \tau_2) \in \mathcal{S}(T_1) \times \mathcal{S}(T_2)} \mathbf{K}'(\tau_1, \tau_2), \end{aligned}$$

where $\mathcal{S}(T)$ denotes a set of subtree patterns in T . The elastic tree kernel is, however, defined as follows:

$$\mathbf{K}(T_1, T_2) = \sum_{(\tau_1, \tau_2) \in \mathcal{M}^{\text{Acc}} \subseteq \mathcal{S}(T_1) \times \mathcal{S}(T_2)} \mathbf{K}'(\tau_1, \tau_2),$$

where by \mathcal{M}^{Acc} we denote a pair of common subtree patterns between T_1 and T_2 mapped by accordant mappings (See Example 5.7). Since this definition is beyond the convolution kernels, it has yet to be shown that the counting function really satisfies the required properties of a kernel function, i.e. positive semidefiniteness. We prove it in the next chapter.

We address these problems in the next chapter.

Example 5.7 Figure 5.3 shows tree mappings counted in the elastic tree kernel. In this example, we focus on a subtree pattern consisting of three nodes. It is difficult to recognize what is the feature space in this kernel. The subtree pattern  is counted in some cases, and not counted in other cases as shown in Figure 5.3. ■

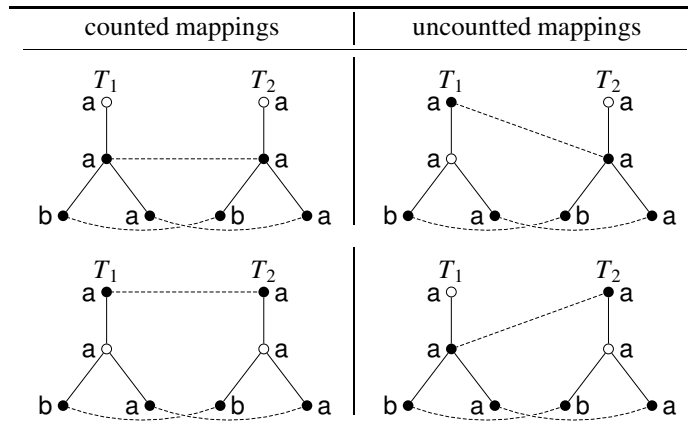


Figure 5.3. What is the feature space in the elastic tree kernel?

5.4.4 String Kernel for Trees

Although tree kernels due to Kashima and Koyanagi [KK02] successfully reduced the computation time by using dynamic programming technique, quadratic computation time is still not sufficient for handling large amounts of data, since the value of this kernel has to be evaluated for most pairs of trees in the training data and test data. Therefore, for more efficient computation, it is important to design a more efficient kernel that is computable in linear time, but with sufficient expressive power.

To tackle this problem, Vishwanathan and Smola proposed a linear-time tree kernel [VS02]. Their idea was to convert trees to strings with brackets generated by preorder traversal, and to count the number of the common substrings among them efficiently by using suffix trees.

This tree kernel does not seem to have enough expressive power for *general-purpose*. Since this kernel considers only for complete subtrees, which can be represented as a substring of the sequence representations, it cannot incorporate internal structure of trees. More specifically, this kernel considers only the subtrees including all their descendant nodes down to the leaves. Therefore, if disjoint labels are assigned to leaves between two trees, the kernel value (i.e., the similarity) ends up with 0, even if these trees are isomorphic except for the labels assigned to leaves. This seems that this kernel does not have enough expressive power, and is a restricted similarity measure between two trees for general purpose.

Example 5.8 Consider two trees in **Figure 5.4**. Filled nodes indicate leaves. **Table 5.2** shows the feature vectors of T_1 and T_2 for Vishwanathan-Smola’s kernel. We have the the following kernel value.

$$\mathbf{K}(T_1, T_2) = \langle (3, 2, 1, 1, 0, 1), (2, 2, 1, 0, 1, 0) \rangle = 11.$$

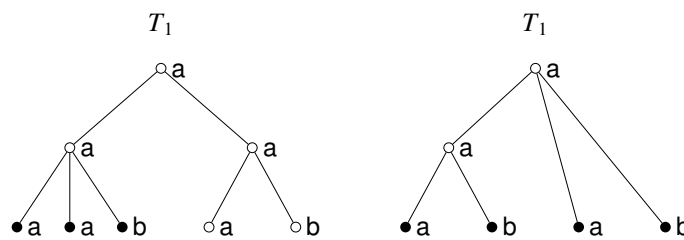


Figure 5.4. Two ordered trees for Vishwanathan-Smola’s tree kernel

Table 5.2. Feature vectors in Vishwanathan-Smola’s tree kernel

T_1	3	2	1	1	0	1
T_2	2	2	1	0	1	0

Example 5.9 For the following two trees T_1 and T_2 as shown in **Figure 5.5**, the liner-time tree kernel gives the value $\mathbf{K}(T_1, T_2) = 0$. In spite of only the one node mismatch at the leaf node between two trees, this kernel regards these two trees as completely different.

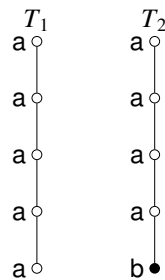


Figure 5.5. Expressive power of the string kernel for trees due to Vishwanathan and Smola

5.5 Summary

In this chapter, we reviewed the convolution kernel as a design framework for learning discrete structures. Also we introduced some tree kernels in prior work, and discussed their properties and raised a few problems to be addressed.

- The elastic tree kernel [KK02] enables to us to deal with ambiguities in trees by extending a convolution kernel for trees. As a result, the elastic tree kernel has turned out to be not a convolution kernel. This means that it is required to prove its positive semidefiniteness.
- The elastic tree kernel has not enough expressive power as originally intended by Kashima and Koyanagi. It is possible to develop a more flexible tree kernel.
- The string kernel for trees due to Vishwanathan and Smola [VS02] is one of the most efficient tree kernels in prior work. But, the expressive power is a restrictive.

In the next chapter, we design novel tree kernels based on the notion of tree mappings, and address the problems raised in this chapter. Chapter 7 is devoted to develop a tree kernel as fast as the string kernel for trees due to Vishwanathan and Smola.



Chapter 6

Mapping Kernels for Trees

In this chapter, we characterize tree kernels based on the class hierarchy of tree mappings established in Chapter 4. We first propose the algorithms for computing counting functions of Tai mappings, alignable mappings, and semi-accordant mappings. We then show that the counting functions of Tai mappings and semi-accordant mappings are actually tree kernels and that these two tree kernels have more flexible expressive power than the elastic tree kernel proposed by Kashima and Koyanagi [KK02]. In contrast, we show that the counting function of alignable mappings is not a tree kernel.

6.1 Recursive Expressions of Counting Functions

In this section, we extend the notion of tree mappings to *forests*, and provide counting functions for the extended tree mappings between two forests.

Remark 6.1 The counting functions in the label tree kernel and the elastic tree kernel both count subtree patterns between two trees, i.e. these require a pair of nodes (s_r, t_r) in the mapping M such that $s_r \geq s$ and $t_r \geq t$ for any $(s, t) \in M$. On the other hand, the counting functions proposed in this chapter count common forest patterns between two trees as well as subtree patterns for generality. It is easy to modify our counting functions in this thesis to count only common subtree patterns by just adding the recurrence Eq.(5.8a).

6.1.1 Mapping-based Similarity between Forests

Recall that the algorithm for Tai distance is in fact defined over two forests as well as two trees. Then we first extend the notion of tree mappings to forests as follows.

Definition 6.1 (Tree Mapping between Forests) Let F_1 and F_2 be two forests. A non-empty mapping $M \subseteq V(F_1) \times V(F_2)$ is said to be a *tree mapping* if and only if M is a tree mapping from $v_1(F_1)$ to $v_2(F_2)$ for two nodes $v_1, v_2 \notin V(F_1) \cup V(F_2)$.

Let $\mathcal{M}^{\mathcal{C}}(F_1, F_2)$ denote the set of all possible \mathcal{C} -mappings between F_1 and F_2 , i.e.

$$\mathcal{M}^{\mathcal{C}}(F_1, F_2) = \{M \mid M \text{ is a tree mapping of class } \mathcal{C} \text{ from } F_1 \text{ to } F_2\}.$$

The counting functions $\mathbf{K}^{\mathcal{C}}(F_1, F_2)$ are defined as follows. First, let us denote a symmetric function by

$$\sigma : \Sigma \times \Sigma \rightarrow \mathbb{R}_0^+,$$

where \mathbb{R}_0^+ denotes the set of all non-negative real numbers. This function σ defines the similarity between labels of nodes. Next, we define the similarity between two trees based on a tree mapping M .

$$\sigma(M) = \prod_{(x_1, x_2) \in M} \sigma(l(x_1), l(x_2)). \quad (6.1)$$

Then the similarity between two forests F_1 and F_2 is defined by

$$\mathbf{K}^C(F_1, F_2) = \sum_{M \in \mathcal{M}^C(F_1, F_2)} \sigma(M). \quad (6.2)$$

6.1.2 Counting Function for Tai Mappings

The recursive expression of the $\mathbf{K}^{\text{Tai}}(T_1, T_2)$ is given as follows.

$$\begin{aligned} \mathbf{K}^{\text{Tai}}(F, \emptyset) &= \mathbf{K}^{\text{Tai}}(\emptyset, F) = 0 \\ \mathbf{K}^{\text{Tai}}(v_1(F'_1) \bullet F''_1, v_2(F'_2) \bullet F''_2) &= \\ &\sigma(l(v_1), l(v_2))(1 + \mathbf{K}^{\text{Tai}}(F'_1, F'_2))(1 + \mathbf{K}^{\text{Tai}}(F''_1, F''_2)) \\ &+ \mathbf{K}^{\text{Tai}}(v_1(F'_1) \bullet F''_1, F'_2 \bullet F''_2) \\ &+ \mathbf{K}^{\text{Tai}}(F'_1 \bullet F''_1, v_2(F'_2) \bullet F''_2) \\ &- \mathbf{K}^{\text{Tai}}(F'_1 \bullet F''_1, F'_2 \bullet F''_2) \end{aligned} \quad (6.3)$$

The following natural properties of Tai mapping (Property 6.2 and Lemma 6.3) play a central role in proving the correctness of Eq.(6.3). We omit the proofs since they are immediately obtained from the definition of Tai mapping (Definition 2.46).

Proposition 6.2 Let M be a Tai mapping from a forest F_1 to a forest F_2 , i.e. $M \subseteq V(F_1) \times V(F_2)$, and F'_i be any subforest of F_i for $i \in \{1, 2\}$. For $M' = M \cap (V(F'_1) \times V(F'_2))$ and $M'' = M \cap (V(F''_1) \times V(F''_2))$, the following hold.

1. If M is a Tai mapping from F_1 to F_2 , then M' is also a Tai mapping from F'_1 to F'_2 .
2. For non-empty $M \subseteq V(F'_1) \times V(F'_2)$, the following two properties are equivalent.
 - (a) M is a Tai mapping from F'_1 to F'_2 .
 - (b) M is a Tai mapping from F_1 to F_2 .

Lemma 6.3 Let F'_i and F''_i be two distinct forests. For a non-empty set $M \subseteq (V(F'_1) \cup V(F''_1)) \times (V(F'_2) \cup V(F''_2))$, the following two conditions are equivalent.

1. $M \cup \{(v_1, v_2)\}$ is a Tai mapping from $v_1(F'_1) \bullet F''_1$ to $v_2(F'_2) \bullet F''_2$.
2. M satisfies the following three conditions.
 - (a) $M = M' \cup M''$.
 - (b) M' is a Tai mapping from F'_1 to F'_2 .
 - (c) M'' is a Tai mapping from F''_1 to F''_2 .

Theorem 6.4 Eq.(6.3) is a counting function for Tai mappings between two trees T_1 and T_2 .

Proof. The left-hand side of Eq.(6.3) is decomposed into the following two disjoint components for $F_1 = v_1(F'_1) \bullet F''_1$ and $F_2 = v_2(F'_2) \bullet F''_2$.

$$\mathbf{K}^{\text{Tai}}(F_1, F_2) = \sum_{M \in \mathcal{M}^{(v_1, v_2)}} \sigma(M) + \sum_{M \in \bar{\mathcal{M}}^{(v_1, v_2)}} \sigma(M),$$

where

$$\begin{aligned} \mathcal{M}^{(v_1, v_2)} &= \{M \in \mathcal{M}^{\text{Tai}}(F_1, F_2) \mid (v_1, v_2) \in M\}, \text{ and} \\ \bar{\mathcal{M}}^{(v_1, v_2)} &= \{M \in \mathcal{M}^{\text{Tai}}(F_1, F_2) \mid (v_1, v_2) \notin M\}. \end{aligned}$$

The set $\mathcal{M}^{(v_1, v_2)} \subseteq \mathcal{M}$ includes all the Tai mappings with (v_1, v_2) whereas the set $\bar{\mathcal{M}}^{(v_1, v_2)} \subseteq \mathcal{M}$ is the complementary of $\mathcal{M}^{(v_1, v_2)}$ in \mathcal{M} , i.e. $\bar{\mathcal{M}}^{(v_1, v_2)} = \mathcal{M} \setminus \mathcal{M}^{(v_1, v_2)}$.

Assume $(v_1, v_2) \in M$. The tree mapping $M \setminus \{(v_1, v_2)\}$ is identical to $M' \cup M''$ for some Tai mappings M' from F'_1 to F'_2 and M'' from F''_1 to F''_2 by Lemma 6.3. On the other hand, for any Tai mappings M' from F'_1 to F'_2 , and M'' from F''_1 to F''_2 , the set $M' \cup M'' \cup \{(v_1, v_2)\}$ is also a Tai mapping from $v_1(F'_1) \bullet F''_1$ to $v_2(F'_2) \bullet F''_2$ by Lemma 6.3.

Hence since $\sigma(M) = \sigma(l(v_1), l(v_2)) \cdot \sigma(M') \cdot \sigma(M'')$, the following holds.

$$\sum_{M \in \mathcal{M}^{(v_1, v_2)}} \sigma(M) = \sigma(l(v_1), l(v_2)) \cdot (1 + \mathbf{K}^{\text{Tai}}(F'_1, F'_2))(1 + \mathbf{K}^{\text{Tai}}(F''_1, F''_2))$$

As for $\bar{\mathcal{M}}^{(v_1, v_2)}$, we decompose it further into the following three components.

$$\bar{\mathcal{M}}^{(v_1, v_2)} = \mathcal{M}^{(-, v_2)} \cup \mathcal{M}^{(v_1, -)} \cup \mathcal{M}^{(-, -)},$$

where

$$\begin{aligned} \mathcal{M}^{(-, v_2)} &= \{M \in \mathcal{M}^{\text{Tai}}(F_1, F_2) \mid (v_1, w) \in M \wedge w \neq v_2\}, \\ \mathcal{M}^{(v_1, -)} &= \{M \in \mathcal{M}^{\text{Tai}}(F_1, F_2) \mid (w, v_2) \in M \wedge w \neq v_1\}, \text{ and} \\ \mathcal{M}^{(-, -)} &= \{M \in \mathcal{M}^{\text{Tai}}(F_1, F_2) \mid (w_1, w_2) \in M \implies w_1 \neq v_1 \wedge w_2 \neq v_2\}. \end{aligned}$$

Note that the set $\mathcal{M}^{(v_1, -)}$ does not include any Tai mapping such that v_1 is paired with a node in F_2 since v_2 is paired with a node other than v_1 . Intuitively, the notation $\mathcal{M}^{(v_1, -)}$ means that the node v_1 is not paired with any node in F_2 in the concerned set of Tai mappings. By condition 2 of Proposition 6.2, the following holds.

$$\begin{aligned} \sum_{M \in \mathcal{M}^{(-, v_2)}} \sigma(M) &= \mathbf{K}^{\text{Tai}}(v_1(F'_1) \bullet F''_1, F'_2 \bullet F''_2) - \mathbf{K}^{\text{Tai}}(F'_1 \bullet F''_1, F'_2 \bullet F''_2), \\ \sum_{M \in \mathcal{M}^{(v_1, -)}} \sigma(M) &= \mathbf{K}^{\text{Tai}}(F'_1 \bullet F''_1, v_2(F'_2) \bullet F''_2) - \mathbf{K}^{\text{Tai}}(F'_1 \bullet F''_1, F'_2 \bullet F''_2), \text{ and} \\ \sum_{M \in \mathcal{M}^{(-, -)}} \sigma(M) &= \mathbf{K}^{\text{Tai}}(F'_1 \bullet F''_1, F'_2 \bullet F''_2). \end{aligned}$$

The correctness of Eq.(6.3) is shown by adding together all of the above components, i.e.

$$\sum_{M \in \bar{\mathcal{M}}^{(v_1, v_2)}} \sigma(M) = \sum_{M \in \mathcal{M}^{(-, v_2)}} \sigma(M) + \sum_{M \in \mathcal{M}^{(v_1, -)}} \sigma(M) + \sum_{M \in \mathcal{M}^{(-, -)}} \sigma(M).$$

■

6.1.3 Template of Counting Function for Subclasses of Tai Mappings

As Proposition 6.2 and Lemma 6.3 do not hold for the subclasses of Tai mapping, the simple Eq.(6.3) is not applicable to the other three classes, i.e. the accordant, semi-accordant or alignable mapping. However, a common template of counting functions exists that is applicable to all three of the subclasses.

$\mathcal{C} \in \{\text{ACCORDANT}, \text{SEMI-ACCORDANT}, \text{ALIGNABLE}\}$	
$\mathbf{K}^{\mathcal{C}}(\emptyset, F) = \mathbf{K}^{\mathcal{C}}(F, \emptyset) = \mathbf{K}_f^{\mathcal{C}}(T, \emptyset) = 0$	(6.6a)
$\mathbf{K}^{\mathcal{C}}(T_1 \bullet F_1, T_2 \bullet F_2) = \mathbf{K}_1^{\mathcal{C}}(T_1 \bullet F_1, T_2 \bullet F_2) + \mathbf{K}^{\mathcal{C}}(F_1, F_2)$	(6.6b)
$\mathbf{K}_f^{\mathcal{C}}(v(F_1), T_2 \bullet F_2) = \mathbf{K}_f^{\mathcal{C}}(v(F_1), T_2) - \mathbf{K}_f^{\mathcal{C}}(T_2, F_1) + \mathbf{K}_f^{\mathcal{C}}(v(F_1), F_2) - \mathbf{K}^{\mathcal{C}}(F_1, F_2) + \mathbf{K}^{\mathcal{C}}(F_1, T_2 \bullet F_2)$	(6.6c)
$\mathbf{K}_f^{\mathcal{C}}(v_1(F_1), v_2(F_2)) = \sigma(l(v_1), l(v_2)) \cdot (1 + \mathbf{K}_2^{\mathcal{C}}(F_1, F_2)) + \mathbf{K}_f^{\mathcal{C}}(v_1(F_1), F_2) + \mathbf{K}_f^{\mathcal{C}}(v_2(F_2), F_1) - \mathbf{K}^{\mathcal{C}}(F_1, F_2)$	(6.6d)

$\mathbf{K}_1^{\mathcal{C}}(F_1, F_2)$ and $\mathbf{K}_2^{\mathcal{C}}(T_1, T_2)$ in the template are defined as follows.

$$\begin{aligned} \mathbf{K}_1^{\mathcal{C}}(T_1 \bullet F_1, T_2 \bullet F_2) &= \sum_{M \in \mathcal{M}_1} \sigma(M), \\ \mathbf{K}_2^{\mathcal{C}}(F_1, F_2) &= \sum_{M \in \mathcal{M}_2} \sigma(M), \end{aligned}$$

where

$$\begin{aligned} \mathcal{M}_1 &= \{M \in \mathcal{M}^C(T_1 \bullet F_1, T_2 \bullet F_2) \mid \\ &\quad M \cap (V(T_1) \times V(T_2 \bullet F_2)) \neq \emptyset \vee M \cap (V(T_1 \bullet F_1) \times V(T_2)) \neq \emptyset\}, \text{ and} \\ \mathcal{M}_2 &= \{M \in \mathcal{M}^C(v_1(F_1), v_2(F_2)) \mid M \cup \{(v_1, v_2)\} \in \mathcal{M}^C(v_1(F_1), v_2(F_2))\}. \end{aligned}$$

The recursive expressions for $\mathbf{K}_1^C(F_1, F_2)$ and $\mathbf{K}_2^C(F_1, F_2)$ are given afterward. $\mathbf{K}_1^C(T_1, T_2)$ takes two trees as the arguments, and $\mathbf{K}_f^C(T, F)$ takes a tree and a forest. Eq.(6.6b) to Eq.(6.6d) are verified in the similar way to Eq.(6.3), and the following Proposition 6.5 is used instead of Proposition 6.2 and Lemma 6.3.

Proposition 6.5 Let \mathcal{C} be one of ACCORDANT, SEMI-ACCORDANT, or ALIGNABLE. Further, let F'_i be a subforest of F_i satisfying one of the following.

- $F_i = F'_i$.
- $F_i = v_i(F'_i)$.
- $F_i = G_i \bullet F'_i \bullet H_i$ for some subforests G_i, H_i of F_i .

Then the following two properties hold.

1. For a \mathcal{C} -mapping M from F_1 to F_2 , $M' = M \cap (V(F'_1) \times V(F'_2))$ is a \mathcal{C} -mapping from F'_1 to F'_2 .
2. For arbitrary $M \subseteq V(F'_1) \times V(F'_2)$, the following two properties are equivalent.
 - (a) M is a \mathcal{C} -mapping from F'_1 to F'_2 ; and
 - (b) M is a \mathcal{C} -mapping from F_1 to F_2 .

In the rest of this section, we use the following notations for $F_i = \bullet_{j=1}^{n_i} T_i^j$ ($i \in \{1, 2\}$) and $M \in \mathcal{M}^C(F_1, F_2)$.

- $M[i, *] = M \cap (V(T_1^i) \times V(F_2))$ for each $i \in \{1, n_1\}$.
- $M[*, j] = M \cap (V(F_1) \times V(T_2^j))$ for each $j \in \{1, n_2\}$.
- $M[i, j] = M \cap (V(T_1^i) \times V(T_2^j)) = M[i, *] \cap M[*, j]$ for each $i \in \{1, n_1\}$ and each $j \in \{1, n_2\}$.

In what follows, we show the specific counting functions according to the class of tree mappings.

Expressions for $\mathbf{K}_1^{\text{Acc}}$ and $\mathbf{K}_1^{\text{Acc}\sharp}$

$$\begin{aligned} \mathcal{C} &\in \{\text{ACCORDANT}, \text{SEMI-ACCORDANT}\} \\ \mathbf{K}_1^C(T_1 \bullet F_1, T_2 \bullet F_2) &= \mathbf{K}_1^C(T_1, T_2) \cdot (\mathbf{K}_3^C(F_1, F_2) - 1) \\ &\quad + \mathbf{K}_f^C(T_1, T_2 \bullet F_2) - \mathbf{K}_f^C(T_1, F_2) \\ &\quad + \mathbf{K}_f^C(T_2, T_1 \bullet F_1) - \mathbf{K}_f^C(T_2, F_1) \\ &\quad + \mathbf{K}^C(T_1 \bullet F_1, F_2) + \mathbf{K}^C(F_1, T_2 \bullet F_2) - 2\mathbf{K}^C(F_1, F_2) \end{aligned} \quad (6.7)$$

The function \mathbf{K}_3^C is defined as

$$\mathbf{K}_3^C(F_1, F_2) = \sum_{M \in \mathcal{M}_3} \sigma(M),$$

where

$$\mathcal{M}_3 = \{M \in \mathcal{M}^C(F_1, F_2) \mid M[i, j] \neq \emptyset \wedge M[p, q] \neq \emptyset \implies (i = p \Leftrightarrow j = q)\}.$$

Any tree mapping in \mathcal{M}_3 consists of tree mappings between isolated subtrees of F_1 and F_2 (See **Figure 6.1**).

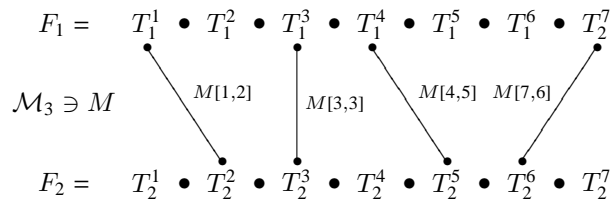


Figure 6.1. A tree mapping in \mathcal{M}_3 : $M = M[1, 2] \cup M[3, 3] \cup M[4, 5] \cup M[7, 6]$

$$\begin{aligned}
\mathcal{C} &\in \{\text{ACCORDANT}, \text{SEMI-ACCORDANT}\} \\
\mathbf{K}_3^{\mathcal{C}}(F, \emptyset) &= \mathbf{K}_3^{\mathcal{C}}(\emptyset, F) = 0 \\
\mathbf{K}_3^{\mathcal{C}}(T_1 \bullet F_1, T_2 \bullet F_2) &= \mathbf{K}_1^{\mathcal{C}}(T_1, T_2) \cdot (1 + \mathbf{K}_3^{\mathcal{C}}(F_1, F_2)) \\
&\quad + \mathbf{K}_3^{\mathcal{C}}(F_1, T_2 \bullet F_2) + \mathbf{K}_3^{\mathcal{C}}(T_1 \bullet F_1, F_2) - \mathbf{K}_3^{\mathcal{C}}(F_1, F_2)
\end{aligned} \tag{6.8}$$

We show the correctness of Eq.(6.7) by Proposition 6.6 and its corollary.

Proposition 6.6 (Decomposition of Accordant Mapping) Let \mathcal{C} be either ACCORDANT or SEMI-ACCORDANT. Let $F_1 = \bullet_{i=1}^{n_1} T_1^i$ and $F_2 = \bullet_{j=1}^{n_2} T_2^j$. For non-empty $M \subseteq V(F_1) \times V(F_2)$, the following are equivalent.

1. M is a \mathcal{C} -mapping.
2. All of the following properties hold.
 - (i) $M[i, *]$ is a \mathcal{C} -mapping for all $i \in \{1, n_1\}$.
 - (ii) $M[* , j]$ is a \mathcal{C} -mapping for all $j \in \{1, n_2\}$.
 - (iii) If $M[i, j] \neq \emptyset$, one of the following types of tree mappings holds.
 - (a) $M = M[i, *]$
 - (b) $M = M[* , j]$
 - (c) If $M[p, q] \neq \emptyset$, then $(i, j) = (p, q)$, $i < p \wedge j < q$, or $i > p \wedge j > q$ holds.

Proof. (1 \Rightarrow 2) The property 1 implies (i) and (ii) by Proposition 6.5. In order to prove (1 \Rightarrow 2), it suffices to derive a contradiction from the assumption that $M \neq M[i, *]$, $M \neq M[* , j]$ and $M[i, q] \neq \emptyset$ for $q \neq j$. From the assumption, there exist $(x_1, x_2) \in M[i, j]$, $(y_1, y_2) \in M[i, q]$ and $(z_1, z_2) \in M[p, q]$ for $p \neq i$, and therefore the following holds in $v_1(F_1)$ and $v_2(F_2)$.

$$x_1 \sim y_1 <_{F_1} x_1 \sim z_1 = v_1 \quad \text{and} \quad v_2 = x_2 \sim y_2 \geq_{F_2} x_2 \sim z_2.$$

Note that any two of x_1, y_1, z_1 are comparable by the hierarchical order, since $x_1, y_1 \in T_1^i$, $z_1 \in T_1^p$, $x_2 \in T_2^j$ and $y_2 \in T_2^q$. This contradicts the hypothesis that M is of \mathcal{C}

(2 \Rightarrow 1) We omit the proof since it is obvious. ■

Corollary 6.7 Let \mathcal{C} be one of ACCORDANT and SEMI-ACCORDANT. For \mathcal{C} -mappings of type (c) in Proposition 6.6, M' from F'_1 to F'_2 and M'' from F''_1 to F''_2 , the tree mapping $M = M' \cup M''$ is a \mathcal{C} -mapping of type (c) from $F'_1 \bullet F''_1$ to $F'_2 \bullet F''_2$.

We assume $M[1, 1] \neq \emptyset$. The following show the contributions of the tree mapping of type (a), (b) and (c) in Proposition 6.6 to $\mathbf{K}_1^{\mathcal{C}}(T_1 \bullet F_1, T_2 \bullet F_2)$.

- (a) $\mathbf{K}_1^{\mathcal{C}}(T_1, T_2 \bullet F_2) - \mathbf{K}_1^{\mathcal{C}}(T_1, F_2) - \mathbf{K}_1^{\mathcal{C}}(T_1, T_2)$
- (b) $\mathbf{K}_1^{\mathcal{C}}(T_2, T_1 \bullet F_1) - \mathbf{K}_1^{\mathcal{C}}(T_2, F_1) - \mathbf{K}_1^{\mathcal{C}}(T_1, T_2)$
- (c) $\mathbf{K}_1^{\mathcal{C}}(T_1, T_2) \cdot (1 + \mathbf{K}_3^{\mathcal{C}}(F_1, F_2))$

If $M[1, 1] = \emptyset$, the contribution to $\mathbf{K}_1^{\mathcal{C}}(T_1 \bullet F_1, T_2 \bullet F_2)$ is given as follows.

$$\mathbf{K}^{\mathcal{C}}(T_1 \bullet F_1, F_2) + \mathbf{K}^{\mathcal{C}}(F_1, T_2 \bullet F_2) - 2\mathbf{K}^{\mathcal{C}}(F_1, F_2)$$

Expressions for K_1^{ALN} and K_2^{Acc}

$$\begin{aligned}
K_1^{\text{ALN}}(\bullet_{i=1}^m T_1^i, \bullet_{j=1}^n T_2^j) &= \kappa_{1,1}^{1,1}(1 + \kappa_{2,n}^{2,m}) \\
&+ \sum_{j=2}^n \left\{ (\kappa_{1,j}^{1,1} - \kappa_{1,j-1}^{1,1})(1 + \kappa_{j+1,n}^{2,m}) + \sum_{i=2}^m (\kappa_{j,j}^{1,i} - \kappa_{j,j}^{1,i-1} - \kappa_{j,j}^{2,i} + \kappa_{j,j}^{2,i-1})(1 + \kappa_{j+1,n}^{i+1,m}) \right\} \\
&+ \sum_{i=2}^m \left\{ (\kappa_{1,1}^{1,i} - \kappa_{1,1}^{1,i-1})(1 + \kappa_{2,n}^{i+1,m}) + \sum_{j=2}^n (\kappa_{1,j}^{i,i} - \kappa_{1,j-1}^{i,i} - \kappa_{2,j}^{i,i} + \kappa_{2,j-1}^{i,i})(1 + \kappa_{j+1,n}^{i+1,m}) \right\} \quad (6.9)
\end{aligned}$$

Let a, b, c , and d satisfy $1 \leq a \leq b \leq n_1$ and $1 \leq c \leq d \leq n_2$.

$$\kappa_{c,d}^{a,b} = K^{\text{ALN}}(\bullet_{i=a}^b T_1^i, \bullet_{j=c}^d T_2^j) \quad (6.10)$$

We show the correctness of Eq.(6.9) by using the following proposition. (Proposition 6.8) and its corollary.

Proposition 6.8 For non-empty $M \subseteq V(\bullet_{i=1}^{n_1} T_1^i) \times V(\bullet_{j=1}^{n_2} T_2^j)$, the following are equivalent.

1. M is an alignable mapping.
2. The following four properties hold.
 - (a) $M[i, *]$ is an alignable mapping for all $i \in \{1, n_1\}$.
 - (b) $M[* , j]$ is an alignable mapping for all $j \in \{1, n_2\}$.
 - (c) If $M[i, j] \neq \emptyset$, $M[p, q] \neq \emptyset$ and $i < p$, then $j \leq q$.
 - (d) None of (i, j) satisfies $M[i, j] \neq \emptyset$, $M[i, *] \setminus M[i, j] \neq \emptyset$ and $M[* , j] \setminus M[i, j] \neq \emptyset$.

Proof. (1 \Rightarrow 2) To prove that (1) implies (2), it suffices to show that M is not alignable if $(x_1, x_2) \in M[i, j]$, $(y_1, y_2) \in M[i, *] \setminus M[i, j]$ and $(z_1, z_2) \in M[* , j] \setminus M[i, j]$ for some (i, j) . Since $x_1, y_1 \in T_1^i$ and $z_1 \notin T_1^i$, $x_1 \sim y_1 < x_1 \sim z_1 = v_1$ holds. In the same way, $y_2 \sim z_2 < x_2 \sim z_2$ holds. Thus M is not alignable.

(2 \Rightarrow 1) To prove that (2) implies (1), it suffices to show that $x_2 \sim z_2 = y_2 \sim z_2$ holds, if $x_1 \sim y_1 < x_1 \sim z_1$ for any $(x_1, x_2), (y_1, y_2), (z_1, z_2) \in M$.

If $x_1, y_1, z_1 \in T_1^i$ for some i , the assertion is derived from (a). Therefore we assume $x_1, y_1 \in T_1^i$ and $z_1 \notin T_1^i$. Further, let $x_2 \in T_2^j, y_2 \in T_2^p$ and $z_2 \in T_2^q$ for j, p, q . If any two of j, p, q are distinct, there is nothing to prove. On the contrary, if $j = p = q$, $x_2 \sim z_2 = y_2 \sim z_2$ is derived from (b). Since $j = p$ and $q \neq j$ hold by (d), $x_2 \sim z_2 = y_2 \sim z_2 = v_2$ holds. \blacksquare

Corollary 6.9 For any alignable mappings, M' from F'_i to F'_i and M'' from F''_i to F''_i , the tree mapping $M = M' \cup M''$ is an alignable mapping from $F'_1 \bullet F''_1$ to $F'_2 \bullet F''_2$.

The entire situation is divided into the following cases.

1. $M[1, *] = M[* , 1] = M[1, 1]$.
2. $M[1, j] \neq \emptyset$ for some $j > 1$ and $M[i, j] = \emptyset$ for any $i > 1$.
3. $M[1, j] \neq \emptyset$ for some $j > 1$ and $M[i, j] \neq \emptyset$ for some $i > 1$.
4. $M[i, 1] \neq \emptyset$ for some $i > 1$ and $M[i, j] = \emptyset$ for any $j > 1$.
5. $M[i, 1] \neq \emptyset$ for some $i > 1$ and $M[i, j] \neq \emptyset$ for some $j > 1$.

By (1) of Proposition 6.5, Proposition 6.8 and Corollary 6.9, the contribution of each case stated above on $K_1^{\text{C}}(\bullet_{i=1}^m T_1^i, \bullet_{j=1}^n T_2^j)$ is calculated as follows.

1. $\kappa_{1,1}^{1,1}(1 + \kappa_{2,n}^{2,m})$
2. $\sum_{j=2}^n (\kappa_{1,j}^{1,1} - \kappa_{1,j-1}^{1,1}) (1 + \kappa_{j+1,n}^{2,m})$

$$\begin{aligned}
3. & \sum_{j=2}^n \sum_{i=2}^m \left(\kappa_{j,j}^{1,i} - \kappa_{j,j}^{1,i-1} - \kappa_{j,j}^{2,i} + \kappa_{j,j}^{2,i-1} \right) \left(1 + \kappa_{j+1,n}^{i+1,m} \right) \\
4. & \sum_{i=2}^m \left(\kappa_{1,1}^{1,i} - \kappa_{1,1}^{1,i-1} \right) \left(1 + \kappa_{2,n}^{i+1,m} \right) \\
5. & \sum_{i=2}^m \sum_{j=2}^n \left(\kappa_{1,j}^{i,i} - \kappa_{1,j-1}^{i,i} - \kappa_{2,j}^{i,i} + \kappa_{2,j-1}^{i,i} \right) \left(1 + \kappa_{j+1,n}^{i+1,m} \right)
\end{aligned}$$

Expressions for $\mathbf{K}_2^{\text{Acc}}$

$$\mathbf{K}_2^{\text{Acc}}(F_1, F_2) = \mathbf{K}_3^{\text{Acc}}(F_1, F_2) \quad (6.11)$$

Eq.(6.11) is a direct corollary of the following proposition (Proposition 6.10).

Proposition 6.10 Let $T_1 = v_1(\bullet_{i=1}^{n_1} T_1^i)$ and $T_2 = v_2(\bullet_{j=2}^{n_2} T_2^j)$. For non-empty $M \subseteq V(\bullet_{i=1}^{n_1} T_1^i) \times V(\bullet_{j=1}^{n_2} T_2^j)$, the following are equivalent to each other.

1. $M \cup \{(v_1, v_2)\}$ is an accordant mapping from T_1 to T_2 .
2. M is an accordant mapping of type (c) from $\bullet_{i=1}^{n_1} T_1^i$ to $\bullet_{j=1}^{n_2} T_2^j$.

Expressions for $\mathbf{K}_2^{\text{Acc}^\#}$ and $\mathbf{K}_2^{\text{ALN}}$

$$\begin{aligned}
& \mathcal{C} \in \{\text{SEMI-ACCORDANT}, \text{ALIGNABLE}\} \\
& \mathbf{K}_2^{\mathcal{C}}(F_1, F_2) = \mathbf{K}^{\mathcal{C}}(F_1, F_2) \quad (6.12)
\end{aligned}$$

Eq.(6.12) is a direct corollary of Proposition 4.22.

Termination

The left-to-right recursive evaluations of Eq. (6.3) to Eq. (6.12) terminate due to the base expression in Eq. (6.6a). Each counting function can be evaluated by dynamic programming as in the case of the algorithms for the tree edit distance. Hence, the time complexities are $O(n^4)$ for Tai, $O(n^2 d^2)$ for the alignable, $O(n^2)$ for the semi-accordant, and $O(n^2)$ for the accordant mapping class, where n denotes the size of trees, and d denotes the maximum degree.

6.2 Positive Semidefiniteness of Counting Functions

We assume that the label-similarity function σ is positive semidefinite. Then intuition may suggest that the positive semidefiniteness of $\mathbf{K}^{\mathcal{C}}$ is inferred from the fact that $\mathbf{K}^{\mathcal{C}}(x, y)$ is represented as a polynomial in $\sigma(a, b)$.

However, this intuition is incorrect when \mathcal{C} is ALIGNABLE. Consider the three forests F_1, F_2 and F_3 , and the label-similarity function σ over $\Sigma = \{a, b, c, d, e, f, g, h\}$ depicted in **Figure 6.2** is a counterexample.

The Gram matrix $[\mathbf{K}^{\text{ALN}}(F_i, F_j)]$ is given by:

$$[\mathbf{K}^{\text{ALN}}(F_i, F_j)] = \begin{bmatrix} 7 + 16\epsilon + 8\epsilon^2 & 7 & 6 \\ 7 & 7 + 8\epsilon & 7 \\ 6 & 7 & 7 + 16\epsilon + 8\epsilon^2 \end{bmatrix}$$

Since its determinant D coincides with $-7 + \epsilon \cdot f_q(\epsilon)$ for some quartic $f_q(\epsilon)$, the determinant D is negative for a sufficiently small $0 < \epsilon < 1$, and therefore, the matrix has at least one negative eigenvalue. This fact means that $\mathbf{K}^{\text{ALN}}(x, y)$ is not a kernel function.

In contrast, $\mathbf{K}^{\mathcal{C}}$ is positive semidefinite when \mathcal{C} is one of TAI, SEMI-ACCORDANT and ACCORDANT (Corollary 6.15). In what follows, we show an important proposition which plays a key role to prove the positive semidefiniteness (Corollary 6.15).

The following notations are used in the next proposition.

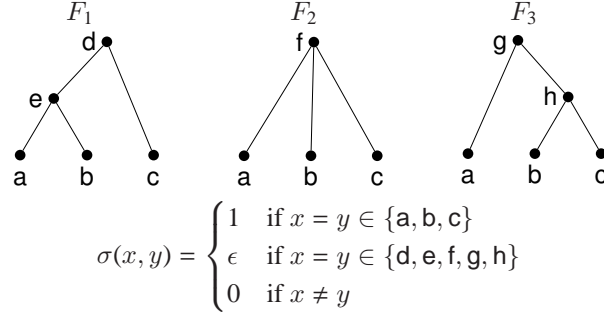


Figure 6.2. A counterexample to positive semidefiniteness

- m, n and d are all positive integers. Further, we assume $\alpha \in \{1, \dots, d\}$, $i, j, a_\alpha \in \{1, \dots, n\}$ and $k_\alpha, l_\alpha, b_\alpha \in \{1, \dots, m\}$.
- \mathcal{I}_m^d is defined as follows.

$$\mathcal{I}_m^d = \{(k_1, \dots, k_d) \in \{1, \dots, m\}^d \mid k_i \leq k_{i+1}\}$$

In addition, for an arbitrary $(k_1, \dots, k_d) \in \{1, \dots, m\}^d$, $\iota(k_1, \dots, k_d)$ denotes a permutation of (k_1, \dots, k_d) such that $\iota(k_1, \dots, k_d) \in \mathcal{I}_m^d$. The α -th element of $\vec{k} \in \mathcal{I}_m^d$ is denoted by $[\vec{k}]_\alpha$.

- When A^{ij} are m -dimensional square matrices parameterized by $(i, j) = \{1, \dots, n\}^2$, A denotes the derived mn -dimensional square matrix $[A^{ij}]_{i,j=1,\dots,n}$: the $(m_i + k, m_j + l)$ -element of A is defined to be the (k, l) -element of A^{ij} , and denoted by A_{kl}^{ij} .
- p is a homogeneous polynomial of degree d in the m^2 variables $X_{11}, X_{12}, \dots, X_{mm}$. Further, assume that p is given a representation of

$$p(X_{11}, X_{12}, \dots, X_{mm}) = \sum_{\vec{k} \in \{1, \dots, m\}^d} \sum_{\vec{l} \in \{1, \dots, m\}^d} c_{\vec{k}, \vec{l}} X_{k_1 l_1} \cdots X_{k_d l_d},$$

where $\vec{k} = (k_1, \dots, k_d)$ and $\vec{l} = (l_1, \dots, l_d)$. Note that such representation of p is not unique.

Proposition 6.11 Let A be a matrix of positive semidefiniteness. If there exists $\bar{c}_{\vec{k}} \in \mathbb{R}$ for each $\vec{k} \in \{1, \dots, m\}^d$ such that $c_{\vec{k}, \vec{l}} = \bar{c}_{\vec{k}} \bar{c}_{\vec{l}}$, then the n -dimensional square matrix $[p(A_{11}^{ij}, \dots, A_{mm}^{ij})]_{i,j=1,\dots,n}$ is also positive semidefinite.

Proof. Since the matrix A is positive semidefinite, there exists mn -dimensional square matrix $B = [B_{kl}^{ij}]$ such that $A = {}^t B B$.

$$\begin{aligned} & p(A_{11}^{ij}, \dots, A_{mm}^{ij}) \\ &= \sum_{\vec{k} \in \{1, \dots, m\}^d} \sum_{\vec{l} \in \{1, \dots, m\}^d} c_{\vec{k}, \vec{l}} \prod_{\alpha=1}^d A_{k_\alpha l_\alpha}^{ij} \\ &= \sum_{\vec{k} \in \{1, \dots, m\}^d} \sum_{\vec{l} \in \{1, \dots, m\}^d} c_{\vec{k}, \vec{l}} \prod_{\alpha=1}^d \left(\sum_{a_\alpha=1}^n \sum_{b_\alpha=1}^m B_{b_\alpha k_\alpha}^{a_\alpha i} B_{b_\alpha l_\alpha}^{a_\alpha j} \right) \\ &= \sum_{\vec{a} \in \{1, \dots, n\}^d} \sum_{\vec{b} \in \{1, \dots, m\}^d} \left(\sum_{\vec{k} \in \{1, \dots, m\}^d} \sum_{\vec{l} \in \{1, \dots, m\}^d} c_{\vec{k}, \vec{l}} \prod_{\alpha=1}^d B_{b_\alpha k_\alpha}^{a_\alpha i} \prod_{\alpha=1}^d B_{b_\alpha l_\alpha}^{a_\alpha j} \right) \\ &= \sum_{\vec{a} \in \{1, \dots, n\}^d} \sum_{\vec{b} \in \{1, \dots, m\}^d} \sum_{\vec{k}, \vec{l} \in \mathcal{I}_m^d} c_{\vec{k}, \vec{l}} \prod_{\alpha=1}^d B_{b_\alpha [\vec{k}]_\alpha}^{a_\alpha i} \prod_{\alpha=1}^d B_{b_\alpha [\vec{l}]_\alpha}^{a_\alpha j} \\ &= \sum_{\vec{a} \in \{1, \dots, n\}^d} \sum_{\vec{b} \in \{1, \dots, m\}^d} \left(\sum_{\vec{k} \in \{1, \dots, m\}^d} \bar{c}_{\vec{k}} \prod_{\alpha=1}^d B_{b_\alpha k_\alpha}^{a_\alpha i} \right) \left(\sum_{\vec{l} \in \{1, \dots, m\}^d} \bar{c}_{\vec{l}} \prod_{\alpha=1}^d B_{b_\alpha l_\alpha}^{a_\alpha j} \right) \end{aligned}$$

This immediately indicates that $[p(A_{11}^{ij}, \dots, A_{mm}^{ij})]_{i,j=1,\dots,n}$ is positive semidefinite. \blacksquare

For a positive integer N , let \mathcal{F}_N denote the set of ordered forests F such that $|F| \leq N$.

A *universal tree* \mathcal{T}_N is an ordered tree with a finite set of nodes, into which each forest $F \in \mathcal{F}_N$ is embedded preserving the hierarchical and sibling orders. When a single order-preserving embedding $e_F : V(F) \rightarrow V(\mathcal{T}_N)$ is assigned to each $F \in \mathcal{F}_N$, a pair of \mathcal{T}_N and the set $\{e_F \mid F \in \mathcal{F}_N\}$ is used as a common numbering scheme of nodes for any $F \in \mathcal{F}_N$.

Let \mathcal{C} denote an arbitrary subclass of Tai mapping, including, but not limited to, TAI, ALIGNABLE, SEMI-ACCORDANT and ACCORDANT. The only restriction imposed on \mathcal{C} is to satisfy

$$\{(v_1, v_1), \dots, (v_n, v_n)\} \in \mathcal{M}^{\mathcal{C}}(F, F)$$

for arbitrary F, n and $v_1, \dots, v_n \in F$. In the above, $\mathbf{K}^{\mathcal{C}}(F_1, F_2)$ denotes the set of the \mathcal{C} -mappings from F_1 to F_2 .

Definition 6.12 (Absorbent Mapping) The tree mapping class \mathcal{C} is said to be

$F[i]$: the i -th tree in F ($i \in [1, |F|]$) *absorbent*,

$F[i]$: the i -th tree in F ($i \in [1, |F|]$) if and only if, for any N , there exists a pair of a universal tree \mathcal{T}_N and a set of embedding $\{e_F \mid F \in \mathcal{F}_N\}$ such that:

$$\forall(F_1 \in \mathcal{F}_N) \forall(F_2 \in \mathcal{F}_N) \forall(M \in V(F_1) \times V(F_2)) \\ [M \in \mathbf{K}^{\mathcal{C}}(F_1, F_2) \iff (e_{F_1} \times e_{F_2})(M) \in \mathbf{K}^{\mathcal{C}}(\mathcal{T}_N, \mathcal{T}_N)].$$

If \mathcal{C} is *transitive*, the inverse of a given \mathcal{C} -mapping and the composition of given \mathcal{C} -mappings are all \mathcal{C} -mappings. In particular, $\mathcal{M}^{\mathcal{C}}(F, F)$ under map composition forms a group.

Theorem 6.13 Let $\sigma : \Sigma \times \Sigma \rightarrow \mathbb{R}_0^+$ be positive semidefinite.

If \mathcal{C} is absorbent and transitive, the function $\mathbf{K}^{\mathcal{C}}|_{\mathcal{F}_N : \mathcal{F}_N \times \mathcal{F}_N \rightarrow \mathbb{R}_0^+}$ is positive semidefinite for an arbitrary N .

Proof. First, the members of $V(\mathcal{T}_N)$ are numbered in preorder. From now on, x_k denotes the k -th node of \mathcal{T}_N , and $x_k[F]$ does $e_F^{-1}(x_k) \in V(F)$. Further, for given $F_i, F_j \in \mathcal{F}_N$, A_{kl}^{ij} denotes $\sigma(l(x_k[F_i]), l(x_l[F_j]))$. Note that defining $\sigma(l(x_k[F_i]), l(x_l[F_j])) = 0$ for $x_k[F_i] = \emptyset$ or $x_l[F_j] = \emptyset$ does not harm the positive semidefiniteness of $[A_{kl}^{ij}]$.

Let \mathcal{I}^d denote $\{(k_1, \dots, k_d) \mid 1 \leq k_1 \leq \dots \leq k_d \leq m\}$, where m denotes $|V(\mathcal{T}_N)|$. For arbitrary $\vec{k}, \vec{l} \in \{1, \dots, m\}^d$, under the notation of

$$M_{\vec{k}, \vec{l}} = \{(x_{k_1}, x_{l_1}), \dots, (x_{k_d}, x_{l_d})\},$$

$c_{\vec{k}, \vec{l}}$ is defined as follows.

$$c_{\vec{k}, \vec{l}} = \begin{cases} 1, & \text{if } \vec{k}, \vec{l} \in \mathcal{I}^d \text{ and } M_{\vec{k}, \vec{l}} \in \mathcal{M}^{\mathcal{C}}(\mathcal{T}_N, \mathcal{T}_N); \\ 0, & \text{otherwise.} \end{cases}$$

Note that an arbitrary $M \in \mathcal{M}^{\mathcal{C}}(F_1, F_2)$ has exactly one instance of (\vec{k}, \vec{l}) such that $M_{\vec{k}, \vec{l}} = M$ and $c_{\vec{k}, \vec{l}} = 1$, since a Tai mapping preserves the preorder.

The following properties are derived from the hypothesis that \mathcal{C} is transitive.

1. $c_{\vec{k}, \vec{k}} = 1$.
2. $c_{\vec{k}, \vec{l}} = c_{\vec{l}, \vec{k}}$.
3. If $c_{\vec{k}, \vec{l}} = c_{\vec{k}, \vec{l}'} = 1$, then $c_{\vec{l}, \vec{l}'} = 1$.

Consequently, \mathcal{I}^d is decomposed into the disjoint union of $\mathcal{I}_1^d, \dots, \mathcal{I}_{\bar{\alpha}_d}^d$, and the following holds.

$$c_{\vec{k}, \vec{l}} = \begin{cases} 1, & \text{if } \vec{k}, \vec{l} \in \mathcal{I}_{\alpha}^d \text{ for some } 1 \leq \alpha \leq \bar{\alpha}_d; \\ 0, & \text{otherwise.} \end{cases}$$

For each I_α^d , define the homogeneous polynomial $p_{\mathcal{T}_\alpha^d}$ of order d by

$$p_{\mathcal{T}_\alpha^d}(X_{11}, \dots, X_{mm}) = \sum_{\vec{k} \in \mathcal{T}_\alpha^d} \sum_{\vec{l} \in \mathcal{T}_\alpha^d} X_{k_1 l_1} \dots X_{k_d l_d}.$$

It is apparent that $p_{\mathcal{T}_\alpha^d}$ satisfies the hypotheses of Proposition 6.11 by defining

$$c_{\vec{k}} = \begin{cases} 1, & \text{if } \vec{k} \in \mathcal{T}_\alpha^d; \\ 0, & \text{otherwise.} \end{cases}$$

Therefore, $[p_{\mathcal{T}_\alpha^d}(A_{11}^{ij}, \dots, A_{mm}^{ij})]$ is positive semidefinite.

Thus, to complete the proof, it suffices to show that the sum of all of $p_{\mathcal{T}_\alpha^d}(A_{11}^{ij}, \dots, A_{mm}^{ij})$ coincides with $\mathbf{K}^{\mathcal{C}}(F_i, F_j)$, and it is apparent, since \mathcal{C} is absorbent by hypothesis. \blacksquare

It is easy to see that ACCORDANT, SEMI-ACCORDANT, ALIGNABLE and TAI classes are all absorbent.

In contrast, the alignable mapping class is not transitive as shown in Proposition 2.58. Therefore, we immediately have the following proposition.

Proposition 6.14 The counting function of alignable mappings is not a kernel.

Thus, Corollary 6.15 gives the main assertion of this chapter.

Corollary 6.15 Let \mathcal{C} be one of TAI, SEMI-ACCORDANT, ACCORDANT. $\mathbf{K}^{\mathcal{C}}|_{\mathcal{F}_N}: \mathcal{F}_N \times \mathcal{F}_N \rightarrow \mathbb{R}_0^+$ is positive semidefinite for an arbitrary N , if and only if $\sigma: \Sigma \times \Sigma \rightarrow \mathbb{R}_0^+$ is positive semidefinite.

Theorem 6.13 has a wide range of applications. For example, the subtree-congruent mapping class is absorbent and transitive. Moreover, let LEAF-TAI, LEAF-SEMI-ACCORDANT and LEAF-ACCORDANT respectively denote the subclasses of TAI, SEMI-ACCORDANT and ACCORDANT such that, for M belonging to the subclasses, x and y are both leaves if $(x, y) \in M$. They are also absorbent and transitive. Therefore, the counting functions for those mapping classes are positive semidefiniteness.

Proof. First, a universal tree \mathcal{T}_N is defined. The universal tree \mathcal{T}_N and an embedding (i.e. an injective map) $e: V(F) \rightarrow V(\mathcal{T}_N)$ are chosen to support the following notations and properties.

- All the nodes of \mathcal{T}_N are numbered so as to satisfy the following conditions.
 - (i) The numbering starts with 1 and is sequential.
 - (ii) If $x < y$, then the number assigned to x is larger than that assigned to y .
 - (iii) If $x \prec y$, then the number assigned to x is smaller than that assigned to y .

Note that such a numbering uniquely exists. x_i denotes the i -th node of \mathcal{T}_N .

- e packs $V(F)$ close leftmost and topmost in $V(\mathcal{T}_N)$ in the following sense.
 - (i) If $x_i \prec x_j$ are children of the same parent and if $x_j \in e(V(F))$, then $x_i \in e(V(F))$.
 - (ii) If $x_j < x_i < x_k$ and if $x_j, x_k \in e(V(F))$, then $x_i \in e(V(F))$.
 - (iii) If F is a tree, $e(\text{root}(F))$ is coincident with x_1 , which is the root of \mathcal{T}_N .
 - (iv) If F is not a tree, x_2 , which is the leftmost child of x_1 , is in $e(V(F))$.
- $x_i[F]$ denotes $e^{-1}(x_i) \in V(F)$.
- $A_{x_i, x_j} = \sigma(l(x_i[F_1]), l(x_j[F_2]))$, for given $F_1, F_2 \in \mathcal{F}_N$.

Let \vec{k} and \vec{l} be non-decreasing serieses of order n . Hence, $k_1 \leq k_2 \leq \dots \leq k_n$ and $l_1 \leq l_2 \leq \dots \leq l_n$ hold for $\vec{k} = (k_1, \dots, k_n)$ and $\vec{l} = (l_1, \dots, l_n)$. Then, $c_{\vec{k}, \vec{l}}$ is defined as follows.

$$c_{\vec{k}, \vec{l}} = \begin{cases} 1, & \text{if } \{(x_{k_1}, x_{l_1}), \dots, (x_{k_n}, x_{l_n})\} \in \mathcal{M}^{\mathcal{C}}(\mathcal{T}_N, \mathcal{T}_N); \\ 0, & \text{otherwise.} \end{cases}$$

Since \mathcal{C} is one of TAI, SEMI-ACCORDANT and ACCORDANT, the following hold.

1. $c_{\vec{k}, \vec{k}} = 1$.
2. $c_{\vec{k}, \vec{l}} = c_{\vec{l}, \vec{k}}$.
3. If $c_{\vec{k}, \vec{l}} = c_{\vec{k}, \vec{l}'} = 1$, then $c_{\vec{l}, \vec{l}'} = 1$.

Note that the property 3 does not holds true, if \mathcal{C} is ALIGNABLE.

As a consequence of the above properties, $\mathcal{I}_1^n, \dots, \mathcal{I}_{\alpha_n}^n$ exist and satisfy the following.

- \mathcal{I}_α^n is a set of non-decreasing serieses of order n .
- $\mathcal{I}_\alpha^n \cap \mathcal{I}_{\alpha'}^n = \emptyset$ for $\alpha \neq \alpha'$
- For any $\vec{k}, \vec{l} \in \mathcal{I}_\alpha^n$, $c_{\vec{k}, \vec{l}} = 1$.
- If $c_{\vec{k}, \vec{l}} = 1$ for non-decreasing serieses \vec{k} and \vec{l} , then $\vec{k}, \vec{l} \in \mathcal{I}_\alpha^n$ for some α .

Therefore, the homogeneous part

$$p_n(X_{11}, \dots, X_{\bar{N}\bar{N}}) = \sum_{k_{[1..N]}} \sum_{l_{[1..N]}} c_{\vec{k}, \vec{l}} X_{k_1 l_1} \dots X_{k_n l_n}$$

is decomposed into the sum of

$$p_{\mathcal{I}_\alpha^n}(X_{11}, \dots, X_{\bar{N}\bar{N}}) = \sum_{\vec{k} \in \mathcal{I}_\alpha^n} \sum_{\vec{l} \in \mathcal{I}_\alpha^n} X_{k_1 l_1} \dots X_{k_n l_n}.$$

Since it is apparent that $p_{\mathcal{I}_\alpha^n}$ satisfies the hypotheses of Proposition 6.11, $p_{\mathcal{I}_\alpha^n}(A_{11}, \dots, A_{\bar{N}\bar{N}})$ defines a positive semidefinite matrix. ■

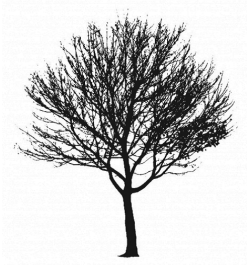
6.3 Summary

A generalization of tree kernels due to Collins and Duffy [CD01], and due to Kashima and Koyanagi [KK02] is addressed in this chapter. Based on the notion of tree mapping, which depicts a common sub-pattern between two trees, it is shown that these existing kernels are the counting functions of tree mappings. By focusing on four major classes of tree mappings proposed in the field of the tree edit distance, four counting functions of tree mappings are proposed according to the four classes. In addition, it is proved that three of the four counting functions are kernel functions, and the other is not by checking their positive semidefiniteness. One of the three tree kernels developed in this chapter turns out to be the elastic tree kernel due to Kashima and Koyanagi. The other two tree kernels are more general than existing tree kernels in the interpretation of common patterns occurring between two trees. We summarize the counting functions of tree mappings for the tree kernels in Table 6.1.

Table 6.1. *Kernels by counting functions of tree mappings*

Tree mapping	Counting function	PSD [†]
Tai mapping [Tai79]	\mathbf{K}^{Tai} (§ 6.1.2)	✓
Alignable mapping [JWZ95]	\mathbf{K}^{ALN} (§ 6.1.3)	
Semi-Accordant mapping [Zha95]	$\mathbf{K}^{\text{Acc}^\#}$ (§ 6.1.3)	✓
Accordant mapping (§ 4.8)	\mathbf{K}^{Acc} (§ 6.1.3), Elastic Tree Kernel [KK02]	✓
Common Subtree Isomorphism [Val98]	Labeled Tree Kernel [KK02]	✓
Bottom-Up Common Subtree Isomorphism [Val98]	String Kernel for Trees [VS02]	✓

[†]PSD stands for positive semidefiniteness.



Chapter 7

Spectrum Kernels for Trees

Most of the existing tree kernels [CD01, KK02, KSK06a] run in quadratic time with respect to the size of the input trees. Also, the kernel function has to be evaluated for most pairs of trees in the training data and test data. As a result, classifiers based on these kernels are too slow for real world applications.

The kernel trick used in their works contributes a significant reduction of computation time, from exponential time for explicit enumeration of common patterns to quadratic time for implicit enumeration by dynamic programming.

Therefore, for more efficient computation, it is important to design an explicitly computable feature vector with low dimension, but with sufficient expressive power.

In this chapter, we propose an expressive and efficient tree kernel based on *tree q -grams*, subtrees isomorphic to paths with q nodes. Note that, by using a linear time algorithm for counting all q -grams in a tree, the tree kernel based on tree q -grams is very efficient for most practical situations. The q -spectrum kernel for trees is identical to the spectrum kernel for strings if strings are given as trees in which every node has at most one child.

In contrast to the tree kernel by Vishwanathan and Smola [VS02] (one of the linear-time kernels), our kernel has enough expressive power to consider the internal structures of trees, and is still computable in linear time.

7.1 Tree q -Grams

In this section, we extend the notion of q -gram for strings [JU91, Ukk92] to trees. Let us begin with introducing a few notions for representing q -grams for ordered labeled trees. Let T be an ordered tree in which each node v_i is indexed by left-to-right postorder numbering. We formulate the *depth sequence* $D(T)$, the *label sequence* $L(T)$, and the *parent sequence* $PS(T)$ of T with n nodes in *left-to-right postorder* as follows.

$$\begin{aligned} D(T) &= \text{dep}(v_1) \cdots \text{dep}(v_n), \\ L(T) &= l(v_1) \cdots l(v_n). \\ PS(T) &= \text{par}(v_1) \cdots \text{par}(v_{n-1}). \end{aligned}$$

We denote $l(v_i)$ by l_i for short. Note that the original depth sequence in [AAK⁺02] has been defined by using *preorder*.

Example 7.1 Consider the tree T in **Figure 7.1** The depth sequence, the label sequence, and the parent sequence of T are given as in **Table 7.1**. ■

For a tree T , and the depth sequence $D = D(T)$, we denote $\max\{d \mid d \in D\}$ by $\max D$. It is obvious that $\text{dep}(T) = \max D$.

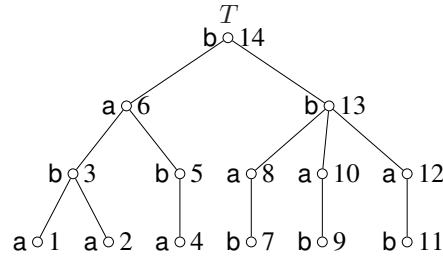


Figure 7.1. An ordered tree with postorder numbering

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
d_i	3	3	2	3	2	1	3	2	3	2	3	2	1	0
l_i	a	a	b	a	b	a	b	a	b	a	b	a	b	b
p_i	3	3	6	5	6	14	8	13	10	13	12	13	14	–

Table 7.1. The depth sequence, the label sequence, and the parent sequence of T .

Note that the reversal of the depth sequence in postorder under left-to-right order of children is the depth sequence in preorder under right-to-left order of children. Then, by using the algorithm PSEQ in Algorithm 7.1, the parent sequence $PS(T)$ of T is obtained from the depth sequence of D in $O(|D|)$ time and $O(\max D)$ space.

Algorithm 7.1 PSEQ

```

procedure PSEQ( $D$ )
  /*  $D$ : a depth sequence in postorder */
   $T[0] \leftarrow |D|$ 
  for  $i = |D| - 1$  downto 1 do
     $PS[i] \leftarrow T[D[i] - 1]$   $T[D[i]] \leftarrow i$ 
  return  $PS$ 

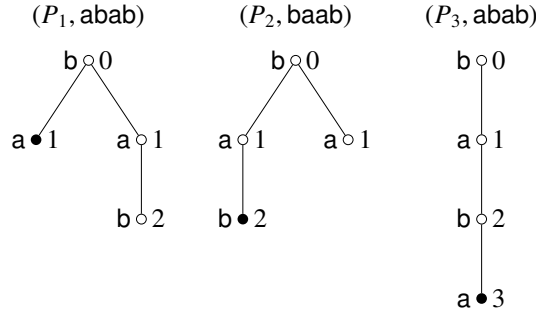
```

We define a *tree q -grams* for trees as a *line tree* consisting of q nodes in which any node has at most two adjacent nodes. For an alphabet Σ , we denote the set of all q -grams by \mathcal{L}_Σ^q . The q -grams have $q - 1$ kinds of isomorphic patterns if the labels are ignored. Then, we divide q -grams into $q - 1$ patterns by the first depth k in its depth sequence (that is, the depth of the left leaf), and denote the patterns by P_k ($1 \leq k \leq q - 1$). We sometimes denote a q -gram by $(P_k, l_1 \cdots l_q)$, which is the pair of a pattern P_k and label sequence $l_1 \cdots l_q \in \Sigma^q$ (See **Figure 7.2**).

Let T and P be trees. Then, we say that P *matches* T at a node v if there exists a one-to-one mapping f from the nodes of P into the nodes of T satisfying the following conditions.

1. f maps the root of P to v .
2. Suppose that f maps x to y and x has children x_1, \dots, x_k from left to right. Then, y has children y_1, \dots, y_m such that $m \geq k$ and there exists a monotone function $g : \{1, \dots, k\} \rightarrow \{1, \dots, m\}$ such that $f(x_i) = y_{g(i)}$ and $g(i_1) < g(i_2)$ whenever $i_1 < i_2$.
3. $l(x) = l(f(x))$ for each $x \in P$.

Also, we say that T has an *occurrence* of P if there exists a node v in T such that P matches T at v . First, we design the algorithm LABELGRAM to count all the q -grams occurring in a given tree as shown in Algorithm 7.2.

Figure 7.2. 4-grams $(P_1, abab)$, $(P_2, baab)$, and $(P_3, abab)$ **Algorithm 7.2** LABELGRAM

```

procedure LABELGRAM( $q, D, L$ )
  /*  $D$ : a depth sequence,  $L$ : a label sequence */
  /* initialize, where  $P[k][w]$  and  $id[k][j]$  are empty. */
   $PS \leftarrow PSEQ(D)$ 
  for  $d = \max D$  downto 0 do
    for  $k = 1$  to  $\min\{q - 1, \max D\}$  do
      if  $k \leq d - q + 1 + 2k \leq \max D$  then
         $count[d] \leftarrow count[d] \cup \{(d - q + 1 + 2k, k)\}$ 
  for  $d = \max D - 1$  downto 1 do
    for  $k = 1$  to  $\max D$  do
      if  $0 \leq d + k \leq \max D$  then
         $shift[d] \leftarrow shift[d] \cup \{(d + k, k)\}$ 
  /* main routine */
  for  $i = 1$  to  $|D| - 1$  do begin
    foreach  $(j, k) \in count[D[i]]$  do begin /* Count */
       $w \leftarrow \varepsilon$ 
      foreach  $l \in id[j][k]$  do begin /* Label */
         $pt1 \leftarrow l; pt2 \leftarrow i$ 
        for  $m = 1$  to  $k$  do
           $w \leftarrow w \cdot L[PS[pt1]]$   $pt1 \leftarrow PS[pt1]$ 
        for  $m = k + 1$  to  $q$  do
           $w \leftarrow w \cdot L[PS[pt2]]$   $pt2 \leftarrow PS[pt2]$ 
         $P[k][w]++$ 
      end /* Label */
    end /* Count */
    if  $D[i] < \max D$  then /* Shift */
      foreach  $(j, k) \in shift[D[i]]$  do
         $id[j][k + 1] \leftarrow id[j][k]$   $id[j][k] \leftarrow \emptyset$ 
         $id[j][k] \leftarrow id[j][k] \cup \{i\}$ 
  end /* for */
  return  $P$ 

```

In order to count all q -grams of an *unlabeled* tree, it is sufficient to count a q -gram P_k with the right leaf $D[i]$ in the algorithm LABELGRAM. This counting requires $O(q|D|)$ time by using two tables *count* and *shift*. On the other hand, in order to count all q -grams of a *labeled* tree, it is necessary to maintain the information of labels in q -grams.

The LABELGRAM scans a given depth sequence from left to right in analogy with a parsing algorithm, and keeps track of all possible occurrences of q -grams as succinct states during scanning. The LABELGRAM employs two tables *count* and *shift*. The table *count* maintains the depth j of the left leaf and the depth d of the right leaf in P_k in order to identify the pattern P_k just from its depths of two leaves. Note that, for generality, we refer to the leaf and the root in pattern P_{q-1} as the left and right nodes respectively. On the other hand, the table *shift* maintains the depth j of the left leaf and the depth d of the root in P_k in order to discard the possibility of the occurrence of P_k .

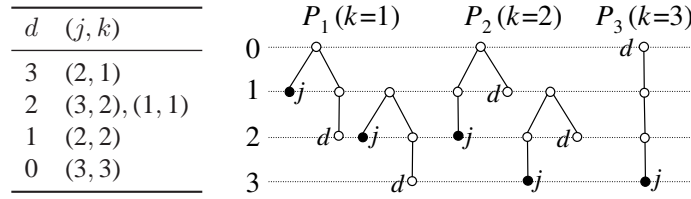


Figure 7.3. The table count for $q = 4$ and $\max D = 3$

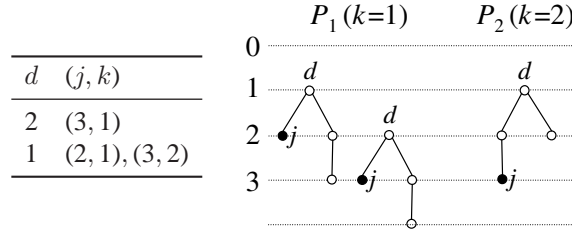


Figure 7.4. The table shift for $q = 4$ and $\max D = 3$

Lemma 7.2 Let P_k be a q -gram ($1 \leq k \leq q - 1$). Then, the following statements hold.

1. If d is the depth of the right leaf (or the root if $k = q - 1$) of P_k , then the depth of the left leaf of P_k is $d - q + 1 + 2k$.
2. If d is the depth of the root of P_k , then the depth of the left leaf of P_k is $d + k$.

Lemma 7.2 guarantees the correctness of the following construction of tables *count* and *shift*: Let D be a depth sequence.

1. For $0 \leq d \leq \max D$, $\text{count}[d]$ consists of the pairs (j, k) such that $j = d - q + 1 + 2k$, $0 \leq j \leq q$ and $1 \leq k \leq q - 1$.
2. For $1 \leq d \leq \max D - 1$, $\text{shift}[d]$ consists of the pairs (j, k) such that $j = d + k$, $0 \leq j \leq \max D$ and $1 \leq k \leq q - 1$.

For example, the tables *count* and *shift* for $q = 4$ and $\max D = 3$ are described in **Figure 7.3** and **Figure 7.4**, respectively.

Here, by ‘.’ we denote the concatenation of two strings. The frequency of the q -gram (P_k, w) is stored in $P[k][w]$, and the index w is stored in $I[k]$. Also $\text{freq}[j][k]$ consists of the pairs (w, f) such that w is a string in Σ with length at most q and f is a positive integer that is the frequency of w , and $\text{label}[d]$ of the triples (k, w, f) such that $1 \leq k \leq q - 1$. In the function $\text{update}(T, \text{Key}, F)$, T is either a table *label* or *freq*, and *Key* is either (k, w) for the table *table*, where we identify $((k, w), f)$ with (k, w, f) , or w for the table *freq*.

The reason why the algorithm LABELGRAM is necessary to maintain the information of labels in q -gram is that a depth sequence is based on the postorder. Even if LABELGRAM finds the depth of the left leaf and the right leaf of some q -gram P_k in “Count” routine, it never finds the labels of the internal nodes in the path from the root to the right leaf in P_k , which we call a *right branch*. Then, the algorithm LABELGRAM finds their labels in the “Label” routine.

Note that every depth of internal nodes in the right branch corresponds to the depth nearest in the right-hand side of the current depth in the depth sequence. Hence, the algorithm LABELGRAM stores the string of labels when it finds some q -gram, where the string consists of the labels in a path from the left leaf to the child of the root, and the right leaf. Furthermore, whenever it finds the depth of internal nodes in the right branch, the “Label” routine in the algorithm LABELGRAM concatenates a label corresponding to the depth to the stored string in the table *label*.

Example 7.3 Consider the trees T_1 and T_2 in Example 7.6 (cf., **Figure 7.5**), where

$$\begin{aligned}
 D(T_1) &= 33232132323210, & D(T_2) &= 32321332323210, \\
 L(T_1) &= aababababababb, & L(T_2) &= abababbabababb.
 \end{aligned}$$

We have already given the tables *count* and *freq* as shown in Figure 7.3 and 7.4. Then, the transitions of $freq[j][k]$ and $label[d]$ for $LABELGRAM(4, D(T_1), L(T_1))$ and $LABELGRAM(4, D(T_2), L(T_2))$ are described in **Table 7.2**. Here, w_f and ${}_k w_f$ denote $(w, f) \in freq[j][k]$ and $(k, w, f) \in label[d]$, respectively.

Also the underlined element w_f in the i -th column of $freq[j][k]$ is added to $label[D[i + 1]]$ by the “Count” routine in the $(i + 1)$ -th iteration of the for-loop. On the other hand, the underlined element ${}_k w_f$ in the i -th column of $label[d]$ is added to $P[k][w \cdot l_{i+1}]$. For example, in the transition of $label[d]$ in Table 7.2, the underlined elements ${}_2 abb_2$ and ${}_1 bab_1$ in $label[2]$ mean that $P[2][abba]$ and $P[1][baba]$ are added to 2 and 1, respectively, because a is the label of the depth 1 in the next column of their elements.

Hence, we obtain the 4-gram profiles of T and S as same as Example 7.6. \blacksquare

Theorem 7.4 The algorithm $LABELGRAM(D(T), L(T), q)$ counts all q -grams occurring in a tree T in $O(q \cdot \deg(T)^2 |T|)$ time and $O(q(\deg(T)^2 + |\Sigma|) |T|)$ space.

Proof. By Lemma 7.2, the following properties of the tables *count* and *shift* hold. (Note the following properties of the tables *count* and *shift*.)

1. $(j, k) \in count[d]$ implies that d and j are the depth of the left and right leaves of P_k .
2. $(j, k) \in shift[d]$ implies that d and j are the depth of the root and the left leaf of P_k .

First, we show the correctness of the algorithm $LABELGRAM$. Since the depth sequence is based on postorder, d_{k+1} and l_{k+1} are the depth and the label of the right leaf of P_k in the depth sequence $D(P_k) = d_1 \cdots d_q$, and the label sequence $L(P_k) = l_1 \cdots l_q$ for $1 \leq k \leq q - 1$ respectively. Furthermore, for P_k , it holds that $d_{i+1} = d_i - 1$ for $k + 1 \leq i \leq q - 1$.

On the other hand, for $(k, w \cdot L[i], f)$ stored in $label[D[i]]$ by the “Count” routine, $w \cdot L[i]$ denotes the label of the q -gram P_k with the right leaf labeled by $L[i]$.

Let $D[j]$ be the current depth. Also suppose that the labels $l_1 \cdots l_i$ ($k + 2 \leq i \leq q - 1$) have been already found in $L(P_k) = l_1 \cdots l_q$ by the “Count” and “Label” routines. Note that the “Label” routine searches for the elements in $label[D[j] + 1]$ and shifts them to $label[D[j]]$. If $d_i = D[j] + 1$, then $D[j]$ is the depth of the parent of the node in P_k labeled by l_i . Hence, $l_{i+1} = L[j]$ and $d_{i+1} = d_i - 1 = D[j]$.

The “Label” routine concatenates every element of $label[D[j] + 1]$ to a label, until the length of such an element is q , so $LABELGRAM$ can count all q -grams with their labels.

Next, we consider the computational complexity of $LABELGRAM$. The size $label[i]$ for each i ($0 \leq i \leq d$) is bounded by the maximum number of q -grams for the node in T with the maximum degree. For a node v with degree $\deg(T)$, the number of q -grams with the root v is bounded by $\deg(T) + (q - 2)\deg(T)(\deg(T) - 1)$, because our q -gram is isomorphic to a line graph, and the number of P_{q-1} is at most $\deg(T)$ and the number of P_k ($1 \leq k \leq q - 2$) is at most $\deg(T) \cdot (\deg(T) - 1)$ as the combination of the left and the right leaves. Then, the size of $label[i]$ is $O(q \cdot \deg(T)^2)$. The “Count” and “Shift” routines in $LABELGRAM$ call just $count[D[i]]$ and $shift[D[i]]$, respectively, both of which sizes are at most $O(q)$ for every i . Also the “Label” routine calls just $label[D[i] + 1]$ and transforms $label[D[i] + 1]$ and $label[D[i]]$, both of which sizes are at most $O(q \cdot \deg(T)^2)$. Hence, the time complexity of $LABELGRAM$ is $(O(q) + O(q \cdot \deg(T)^2)) |T| = O(q \cdot \deg(T)^2 |T|)$.

The size of *count* and *shift* is $O(qd)$ and the size of *label* is $O(qd \cdot \deg(T)^2)$. Also the size of *freq* is $O(qd|\Sigma|)$, because the maximum number of $freq[j][k]$ is the number of different labels, that is, $O(|\Sigma|)$. Furthermore, since the number of all q -grams with the root as a fixed node is $O(q \cdot \deg(T)^2)$, the number of all q -grams in T is $O(q \deg(T)^2 |T|)$, which is the size of P . Then, the space complexity of $LABELGRAM$ is $O(q(d + d|\Sigma| + d \deg(T)^2 + \deg(T)^2 |T|)) = O(q(\deg(T)^2 + |\Sigma|) |T|)$, since $d \leq |T|$ and $q \leq |T|$. \blacksquare

In our experiments, the running time of this algorithm was almost on the order of $O(|T|)$ since the degree of trees is bounded and q is constant.

7.2 Spectrum Kernel for Trees

We first formulate the q -gram profile for trees as in the case of strings. Let T be a tree and $P = (P_k, w) \in \mathcal{L}_\Sigma^q$ a q -gram, where $w \in \Sigma^q$. We denote the total number of the occurrences of (P_k, w) in T by $\#T[(P_k, w)]$ for $1 \leq k \leq q - 1$. Then, the q -gram profile of T is the vector $G_q(T) = (\#T[P])_{P \in \mathcal{L}_\Sigma^q}$.

Now we are ready to present a new tree kernel as a similarity measure between two trees.

Table 7.2. The transition of $\text{freq}[j][k]$ and $\text{label}[d]$ in T_1 and in T_2

j	k	3a	3a	2b	3a	2b	1a	3b	2a	3b	2a	3b	2a	1b	0b
3	1	a_1	a_2		a_1			b_1		b_1				b_1	
3	2			ab_2	ab_2	ab_3			ba_1	ba_1	ba_2	ba_2	ba_3	ba_3	
3	3						aba_3	aba_3	aba_3	aba_3	aba_3	aba_3	aba_3	aba_3	aba_3
<hr/>															
<i>freq[j][k]</i>															
2	1			b_1	b_1	b_2									
2	2						ba_2	ba_2	ba_2	a_1	a_1	a_2	a_2	a_3	ba_2
1	1						a_1	a_1	a_1	a_1	a_1	a_1	a_1	a_1	a_1
<hr/>															
T_1															
d		3a	3a	2b	3a	2b	1a	3b	2a	3b	2a	3b	2a	1b	0b
3				b_1					$1ab_1$						
2				$2abb_2$		$1bab_1$		$1aa_1$	$1aa_1$	$1aa_2$	$1aa_2$	$1aa_3$	$2baa_1$	$2baa_3$	
1										$1aba_1$	$1aba_1$	$1aba_3$			
0														$2bab_2$	$1aab_3$
<hr/>															
<i>label[d]</i>															
3	1	a_1			a_1			b_1							
3	2		ab_1		ab_1	ab_2			ba_2	ba_2	b_1	ba_3	ba_3	ba_4	
3	3					aba_2	aba_2	aba_2	aba_2	aba_2	aba_2	aba_2	aba_2	aba_2	aba_2
<hr/>															
<i>freq[j][k]</i>															
2	1		b_1	b_1	b_2		ba_2	ba_2	ba_2	a_1	a_1	a_2	a_2	a_3	ba_2
2	2						ba_2	ba_2	ba_2	ba_2	ba_2	ba_2	ba_2	ba_2	ba_2
1	1						a_1	a_1	a_1	a_1	a_1	a_1	a_1	a_1	a_1
<hr/>															
T_2															
d		3a2b	3a	2b	1a	3b	3b	2a	3b	2a	3b	2a	1b	0b	
3				$1ba_1$					$1ab_1$						
2				$1bab_1$	$2abb_1$				$1aa_1$	$1aa_1$	$1aa_2$	$1aa_2$	$1aa_3$		
1										$2baa_2$	$2baa_2$	$2baa_3$	$2baa_5$		
0										$1aba_1$	$1aba_1$	$1aba_3$		$2bab_2$	$1aab_3$
<hr/>															
<i>label[d]</i>															
3															
2															
1															
0														$3abab_2$	$3bab_4$

Definition 7.5 (q -Spectrum Kernel) Let T_1 and T_2 be trees. Then, the q -spectrum kernel of T_1 and T_2 is the inner product of $G_q(T_1)$ and $G_q(T_2)$ as follows:

$$K_q(T_1, T_2) = \langle G_q(T_1), G_q(T_2) \rangle$$

Table 7.3. The 4-gram profiles of T_1 (left) and T_2 (right)

P_1	aabb	3	$\langle 6, 8 \rangle, \langle 6, 10 \rangle, \langle 6, 12 \rangle$	P_1	aabb	3	$\langle 5, 8 \rangle, \langle 5, 10 \rangle, \langle 5, 12 \rangle,$
	abab	3	$\langle 8, 9 \rangle, \langle 8, 11 \rangle, \langle 10, 11 \rangle$		abab	3	$\langle 8, 9 \rangle, \langle 8, 11 \rangle, \langle 10, 11 \rangle$
	baba	1	$\langle 3, 4 \rangle$		baba	1	$\langle 2, 3 \rangle$
P_2	abba	2	$\langle 1, 5 \rangle, \langle 2, 5 \rangle$	P_2	abba	1	$\langle 1, 4 \rangle$
	baab	3	$\langle 7, 10 \rangle, \langle 7, 12 \rangle, \langle 9, 12 \rangle$		baab	5	$\langle 6, 10 \rangle, \langle 6, 12 \rangle, \langle 7, 10 \rangle, \langle 7, 12 \rangle, \langle 9, 12 \rangle$
	babb	2	$\langle 3, 13 \rangle, \langle 5, 13 \rangle$		babb	2	$\langle 2, 13 \rangle, \langle 4, 13 \rangle$
P_3	abab	3	$\langle 1, 14 \rangle, \langle 2, 14 \rangle, \langle 4, 14 \rangle$	P_3	abab	2	$\langle 1, 14 \rangle, \langle 3, 14 \rangle$
	babb	3	$\langle 7, 14 \rangle, \langle 9, 14 \rangle, \langle 11, 14 \rangle$		babb	4	$\langle 6, 14 \rangle, \langle 7, 14 \rangle, \langle 9, 14 \rangle, \langle 11, 14 \rangle$

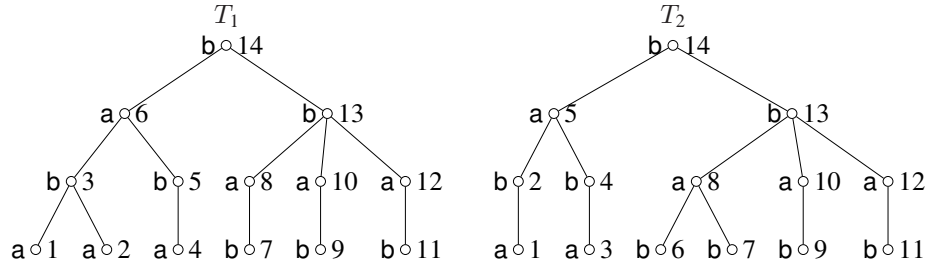


Figure 7.5. The trees T_1 and T_2 in Example 7.6

Example 7.6 Let $\Sigma = \{a, b\}$ and consider the 4-gram profiles of the trees T_1 and T_2 shown in Figure 7.5. Here, the numbers to the right of nodes describe the postorder. Then Table 7.3 denotes (P_i, w) such that $\#T_1[(P_i, w)] > 0$ and $\#T_2[(P_i, w)] > 0$ in the 4-gram profile and their values, respectively. Here, $\langle u, v \rangle$ denotes the path from u to v in T_1 and T_2 . Hence, the following statement holds.

$$\begin{aligned} \mathbf{K}_4(T_1, T_2) &= \langle (3, 3, 1, 2, 3, 2, 3, 3), (3, 3, 1, 1, 5, 2, 2, 4) \rangle \\ &= 58. \end{aligned}$$

■

7.3 q -Gram Distance for Trees

We deviate here from tree kernels, and consider a distance measure based on tree q -grams. As in the case of strings, we can define q -gram distance for trees as follows.

Definition 7.7 (q -Gram Distance) Let T_1 and T_2 be ordered labeled trees, and let q be a fixed natural number. The q -gram distance between T_1 and T_2 is defined as follows:

$$\mathbf{D}_q^{\text{GRAM}}(T_1, T_2) = \|G_q(T_1) - G_q(T_2)\|_1 = \sum_{i=1}^{q-1} \sum_{w \in \Sigma^q} |\#T_1[(P_i, w)] - \#T_2[(P_i, w)]|.$$

We remark that the q -gram distance is not a metric but a pseudometric in the mathematical sense since it may be $\mathbf{D}_q(T_1, T_2) = 0$ even if $T_1 \neq T_2$.

Example 7.8 Let $\Sigma = \{a, b\}$, and consider the 4-gram profiles of the trees T_1 and T_2 described in Figure 7.5.

$$D_4^{\text{GRAM}}(T_1, T_2) = \|(3, 3, 1, 2, 3, 2, 3, 3) - (3, 3, 1, 1, 5, 2, 2, 4)\|_1 = 5.$$

A theoretical property of q -gram distance has yet to be well analyzed in order to apply this distance measure to an efficient filtration of tree edit distance.

7.4 Gram Distribution Kernel

We also present another tree kernel based on tree q -grams for smoothing the kernel values from various q to interpolate the patterns of various lengths. A gram distribution kernel is a generalization of the spectrum tree kernel for this purpose.

By $\text{maxpath}(T)$ we denote the number of nodes on the longest unique path in T . Assume $m = \text{maxpath}(T)$. Then, the *gram distribution* $\mathcal{G}(T)$ of T is the following sequence of all q -gram profiles for every q ($1 \leq q \leq m$).

$$\mathcal{G}(T) = (G_1(T), \dots, G_m(T)).$$

Definition 7.9 (Gram Distribution Kernel) Let T_1 and T_2 be trees, and $m = \min\{\text{maxpath}(T_1), \text{maxpath}(T_2)\}$. Then, the *gram distribution kernel* of T_1 and T_2 is the sum of the inner products of q -gram profiles of T_1 and T_2 for all the possible q as follows:

$$K(T_1, T_2) = \sum_{q=1}^m \langle G_q(T_1), G_q(T_2) \rangle.$$

The algorithm LABELGRAMDIST in Algorithm 7.3 can compute the gram distribution of a tree from its depth sequence D , label sequence L and parent sequences PS , which is the main advantage in using the gram distribution kernel.

The notations in the algorithm LABELGRAMDIST are almost the same as LABELGRAM. The frequency of the q -gram (P_k^q, w) is stored in $P[q][k][w]$. Also, $\text{freq}[j][k]$ consists of the pairs (w, f) such that w is a string over Σ with length at most q and f is a positive integer that is the frequency of w . In the function $\text{update}(T, v, F)$, T is an element of a table freq , and v is a string. Note that $\text{maxpath}(T) \leq 2\text{dep}(T) + 1$. In the algorithm LABELGRAMDIST, we assume that $2\text{dep}(T) + 1 < |T|$.

The main difference between the algorithms LABELGRAMDIST and LABELGRAM is the construction of the labels of the right branch. In LABELGRAM, we have adopted the table *label* and constructed the labels of the right branch, with running the main routine. In contrast, whenever LABELGRAMDIST finds some q , it computes the labels of the right branch for the found q -gram, by using the parent sequence.

For each node v in a tree T rooted at r , let $UP_r(v)$ denote the unique path from v to r . In particular, $UP_r(r) = \{r\}$. In order to compute the value of q and rb in the ‘‘Count’’ routine, we use the following relationship among d, j, k and $|UP_r(p)|$ for a q -gram P_k^q .

1. For some q , let P_k^q be a q -gram ($1 \leq k \leq q - 1$), and let d be the depth of the right leaf of P_k^q and j the depth of the left leaf of P_k^q . Then, it holds that $q = d + 2k + 1 - j$ (Figure 7.6 (left)).
2. For some q , let P_k^q be a q -gram ($1 \leq k \leq q - 1$), and let d be the depth of the right leaf of P_k^q and j the depth of the left leaf of P_k^q . Furthermore, let r and p be the root and the parent node of the right leaf of P_k^q , respectively. It thus holds that $rb(= |UP_r(p)|) = d - j + k$ (Figure 7.6 (right)).

Let T be a tree and $m = \text{maxpath}(T)$. Then, we denote the number of different q -grams occurring in T by $N_{T,q}$ and $\sum_{q=2}^m N_{T,q}$ by N_T . Note that

$$N_{T,q} \leq \min\{|T|, |\Sigma|^q, \text{deg}(T)^q\} \quad \text{and} \quad N_T \leq \min\{|T|, |\Sigma|^m, \text{deg}(T)^m\}.$$

Theorem 7.10 Let D and L be the depth and label sequences of T , respectively. Also suppose that $h = \text{dep}(T)$. Then, the algorithm LABELGRAMDIST computes the gram distribution of T in $O(h^2 N_T |D|)$ time and in $O(N_T + h^2 + |D|)$ space, by traversing D twice.

Algorithm 7.3 LABELGRAMDIST

```

procedure LABELGRAMDIST( $D, L, PS$ )
/*  $D$  : a depth sequence,  $L$  : a label sequence,  $PS$  : a parent sequence */
/* initialize, where  $2 \max D + 1 < |D|$  */
for  $d = \max D - 1$  downto 1 do
  for  $k = 1$  to  $\max D$  do
    if  $0 \leq d + k \leq \max D$  then
       $shift[d] \leftarrow shift[d] \cup \{(d + k, k)\}$ 
   $PL \leftarrow parent\_list(D)$ 
for  $i = 1$  to  $|D|$  do begin
   $P[1][0][L[i]]++$ 
  for  $j = \max D$  downto 1 do begin /* Count */
    for  $k = 1$  to  $j$  do
      foreach  $(w, f) \in freq[j][k]$  do
         $q \leftarrow D[i] + 2k + 1 - j$   $rb \leftarrow D[i] - j + k$   $s \leftarrow \varepsilon$   $pt \leftarrow i$ 
        for  $m = 0$  to  $rb$  do begin /* Label */
           $s \leftarrow s \cdot L[pt]$   $pt \leftarrow PS[pt]$ 
        end /* Label */
         $w \leftarrow w \cdot s$   $P[q][k][w] \leftarrow P[q][k][w] + f$ 
      end /* Count */
      if  $D[i] = 0$  then break
      if  $D[i] < \max D$  then
        foreach  $(j, k) \in shift[D[i]]$  do begin /* Shift */
          foreach  $(w, f) \in freq[j][k]$  do
             $update(freq[j][k + 1], w \cdot L[i], f)$   $freq[j][k] \leftarrow freq[j][k] - \{(w, f)\}$ 
          end /* Shift */
         $update(freq[D[i]][1], L[i], 1)$ 
      end
    return  $P$ 

function  $update(T, v, F)$ 
/*  $T$  : an element of  $freq[ ][ ]$ ,  $v$  : string */
if  $\exists (w, f) \in T$  s.t.  $f > 0$  then  $F \leftarrow f + F$  else  $T \leftarrow T \cup \{(v, F)\}$ 

```

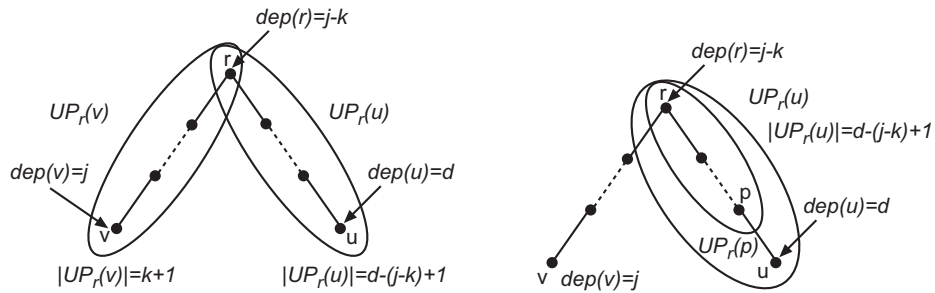


Figure 7.6. The relationship among d, j, k and $|UP_r(p)|$ for a q -gram P_k^q

Proof. In order to show the correctness of LABELGRAMDIST, it is sufficient to show the correctness of the “Count” routine, by using the correctness of GRAMDIST of computing the gram distribution for unlabeled trees.

Due to the relationship among d, j, k and $|UP_r(p)|$ for a q -gram P_k^q , for every $(w, f) \in \text{freq}[j][k]$, LABELGRAMDIST correctly computes q and rb , respectively. Let P_k^q be the found q -gram. Then, w is a string of the left branch of P_k^q . Since rb is the number of nodes in the right branch of P_k^q , the “Label” routine finds the string s concatenating the label of right leaf (in the case that $m = 1$) to the labels of the right branch (in the case that $2 \leq m \leq rb$) in postorder, with traversing the ancestors of the right leaf by using the parent sequence PS . Hence, $w \cdot s$ is the label sequence of P_k^q .

Next, consider the computational complexity of LABELGRAMDIST. The size of the table *shift* is $O(h^2)$. Also the size of the table *freq* is $O(N_T)$. Furthermore, the size of PL is $O(|D|)$. Finally, since $\text{maxpath}(T) \leq 2h+1$, the size of P is $O(N_T)$. Hence, the space complexity of LABELGRAMDIST is $O(h^2) + O(N_T) + O(|D|) = O(N_T + d^2 + |D|)$.

On the other hand, since rb is bounded by h , the time complexity of the “Label” routine is $O(h)$. Since the size of $\text{freq}[j][k]$ is $O(N_T)$, the time complexity of the “Count” routine is $O(h^2 N_T)$. Since the “Shift” routine calls just $\text{shift}[D[i]]$ of which size is $O(h)$, the time complexity of the “Shift” routine is $O(h N_T)$. Since the time complexity of the initialization is $O(h^2 + |D|)$, the time complexity of LABELGRAMDIST is $O(h^2 + |D|) + O((h^2 N_T + h N_T)|D|) = O(h^2 N_T |D|)$.

Finally, the algorithm LABELGRAMDIST traverses D twice, that is, the construction of the parent sequence and the main routine. ■

The algorithm runs in $O(h^2 |T|^2)$ time and $O(h^2 + |T|)$ space since $N_{T,q} \leq \min\{|T|, |\Sigma|^q, \text{deg}(T)^q\}$ and $N_T \leq \min\{|T|, |\Sigma|^m, \text{deg}(T)^m\}$ for $m = \text{maxpath}(T)$. In our experiments, the running time of this algorithm was on the order of $O(|T|^2)$.

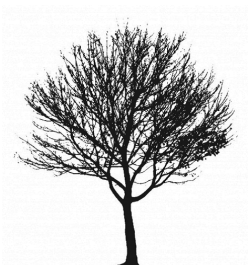
Example 7.11 Consider the tree T in Figure 7.1. Then, the transition of the table *freq* in the algorithm LABELGRAMDIST is described in **Table 7.4**. Here, $(w, f)_v^{q,r}$ denotes that f is the frequency, v is the depth sequence of the q -gram, and r is the number of nodes in the right branch of the q -gram. The underline part of v is corresponding to the string s in the algorithm LABELGRAMDIST, that is, the labels of the right branch. In contrast, w is the labels of the left branch. Note that v, q and r are not stored in the table *freq*. ■

7.5 Summary

In order to develop a fast tree kernel with a sufficient expressive power for practical use, we extend the notion of q -gram for strings to trees. We employ a very simple form of subtrees with q nodes as q -grams, and show an efficient algorithm for counting tree q -gram occurring in a tree by using dynamic programming. Based on the number of q -grams, we define a spectrum tree kernel, and a q -gram based distance measure between trees. Moreover, we propose a gram distribution kernel as a generalization of the spectrum tree kernel. In the next chapter, we demonstrate the effectiveness of these tree kernels by applying them to biological data.

Table 7.4. *The transition of the table freq in T.*

i	1	2	3	4	5	6	7	
p_i	3	3	6	5	6	14	8	
j	k	3a	3a	2b	3a	2b	1a	3b
3	1	$(a, 1)_{aab}^{3,1}$	$(a, 2)_{ab}^{2,0}$		$(a, 1)_{ab}^{2,0}$			$(b, 1)_{ba}^{2,0}$
3	2			$(ab, 2)_{ababa}^{5,2}$	$(ab, 2)_{abba}^{4,1}$	$(ab, 3)_{aba}^{3,0}$		
3	3						$(aba, 3)_{abababb}^{7,3}$	$(aba, 3)_{abaabb}^{6,2}$
2	1			$(b, 1)_{baba}^{4,2}$	$(b, 1)_{bba}^3$	$(b, 2)_{ba}^{2,1}$		
2	2						$(ba, 2)_{bababb}^{6,3}$	$(ba, 2)_{baabb}^{5,2}$
1	1						$(a, 1)_{ababb}^{5,3}$	$(a, 1)_{aabb}^{4,2}$
i	8	9	10	11	12	13	14	
p_i	13	10	13	12	13	14	—	
j	k	2a	3b	2a	3b	2a	1b	0b
3	1		$(b, 1)_{ba}^{2,0}$		$(b, 1)_{ba}^{2,0}$			
3	2	$(ba, 1)_{babab}^{5,2}$	$(ba, 1)_{baab}^{4,1}$	$(ba, 2)_{babab}^{5,2}$	$(ba, 2)_{baab}^{4,1}$	$(ba, 3)_{bab}^{3,0}$		
3	3	$(aba, 3)_{abababb}^{7,3}$	$(aba, 3)_{abaabb}^{6,2}$	$(aba, 3)_{abababb}^{7,3}$	$(aba, 3)_{abaabb}^{6,2}$	$(aba, 3)_{ababb}^{5,1}$	$(aba, 3)_{abab}^{4,0}$	$(bab, 3)_{babb}^{4,0}$
2	1	$(a, 1)_{abab}^{4,2}$	$(a, 1)_{aab}^{3,1}$	$(a, 2)_{abab}^{4,2}$	$(a, 2)_{aab}^{3,1}$	$(a, 3)_{ab}^{2,0}$		
2	2	$(ba, 2)_{bababb}^{6,3}$	$(ba, 2)_{baabb}^{5,2}$	$(ba, 2)_{bababb}^{6,3}$	$(ba, 2)_{baabb}^{5,2}$	$(ba, 2)_{babb}^{4,1}$	$(ba, 2)_{bab}^{3,0}$	
2	2						$(ab, 3)_{abb}^{3,0}$	
1	1	$(a, 1)_{ababb}^{5,3}$	$(a, 1)_{aabb}^{4,2}$	$(a, 1)_{ababb}^{5,3}$	$(a, 1)_{aabb}^{4,2}$	$(a, 1)_{abb}^{3,1}$	$(a, 1)_{ab}^{2,0}$	$(b, 1)_{bb}^{2,0}$



Chapter 8

Application to Glycan Classification

In the previous chapter, we have proposed two novel tree kernels: the *spectrum tree kernel* and the *gram distribution kernel*. The spectrum tree kernel is a natural extension of the spectrum string kernel based on the notion of tree q -gram, and the gram distribution kernel is a sum of spectrum kernels with varying values of q .

In this chapter, we evaluate the effectiveness of these two kernels by empirically comparing their computation time and predictive performance in a glycan structure classification problem with the existing tree kernels.

8.1 Glycan Data

Glycans or sugar chains [Var02] are defined as the third major class of biomolecules next to DNA and proteins. They are polysaccharide structures, or carbohydrate structures, often forming tree structures, as opposed to the linear structure of DNA and proteins. They are known to be extremely crucial for the development and function of multi-cellular organisms as they are found mainly on the cell surface and recognized by various agents to signal a wide variety of events. Only in recent years, however, has bioinformatics focused on glycans, mainly because of the complexity in developing high-throughput techniques to characterize their structures. Databases have been developed for the public to freely search and browse carbohydrate structures, with KEGG/GLYCAN [HGK⁺06], the Consortium for Functional Glycomics, and the German Cancer Research Center (glycosciences.de) [LBLL⁺06] leading the way. From this data, we are now able to mine for structural features that may not be readily clear to the naked eye. In fact, several probabilistic models have been developed recently to attempt to mine such patterns [AKUMK06, HAKU⁺06].

For glycan structures, Hizukuri *et al.* [HYN⁺05] proposed a glycan-specific kernel, to which we refer as the *layered trimer kernel*. This kernel is designed according to the characteristic mechanisms of glycan recognition. In the kernel, the feature space is defined as the set of all the trimers in each layer, and the attribute value is the number of occurrences of trimers weighted according to the significance of components. By using the SVM with the layered trimer kernel, Hizukuri *et al.* successfully showed the effectiveness of their approach by the classification of blood components, and they extracted leukemia specific glycan motifs in humans computationally for the first time.

The trees we deal with in our kernel are node-labeled trees while the structure of a glycan is abstractly represented as a tree consisting regarding single sugars as nodes and the covalent bonds between them as edges, i.e. the nodes and edges are labeled. Therefore we regard each edge label as the prefix of the label assigned to the node right under the edge. We have also incorporated information indicating root and leaf nodes.

In this chapter, we compare our tree kernel with the tree kernel due to Kashima and Koyanagi [KK02] and the layered trimer kernel due to Hizukuri *et al.* [HYN⁺05] in supervised classification problems. Concretely, we deal with glycan structure classification problems in the field of bioinformatics. Classification of glycan structures are a fairly important task since their functions largely depend on their structures.

In our experiments, we use comparable glycan data to Hizukuri *et al.* [HYN⁺05]. We have retrieved glycan structures from the KEGG/GLYCAN database [HGK⁺06] and their annotations from the CarbBank/CCSD database [DA92]. We employ the data set of four blood components, *leukemic cells*, *erythrocyte*, *serum*, and *plasma*, as the class labels according to Hizukuri *et al.* [HYN⁺05].

We also test our method on a different set of data to assess the generality of our method for extracting glycan markers. Because of the larger amount of data and research that have been put into cystic fibrosis, we select data related to this disease. Cystic fibrosis is one of the most lethal genetic disorders in Caucasians, characterized by the production of excessive amounts of viscous mucus secretions in the airways of patients, leading to airway obstruction, chronic bacterial infections, and respiratory failure. Previous studies indicated that CF-derived airway mucins are glycosylated and sulfated differently compared with mucins from nondiseased (ND) individuals [XRD⁺05]. In order to obtain the those structures that were related to CF, we have extracted those entries that are annotated with the word “cystic,” “bronch,” and “respir” as substrings. We have found that there are a sufficient number of structures representing each of these groups to test our method. We have summarized the data used in our experiments in Table 8.1. Note that the total numbers in the table are not the sum of each number of data since these data are overlapping.

Table 8.1. *The data labels, and the number of each data set in the experiments*

leukemia	erythrocyte	plasma	serum	total
191	274	144	202	480
cystic fibrosis	respiratory mucin	bronchial mucin		total
53	123	110		153

8.2 Experimental Results

8.2.1 Computation Time

Figure 8.1 describes the running time for computing the tree q -spectrum kernel \mathbf{K}_q for $2 \leq q \leq 8$, and the labeled ordered tree kernel \mathbf{K}_t proposed by Kashima and Koyanagi [KK02]. Here, the “computation time” is the average time for computing each tree kernel function between all combinations of pairs of trees, after randomly generating 10 trees with the size, degree, and size of alphabet at most 1000, 5, and 8 respectively. Figure 8.1 shows that our kernel \mathbf{K}_q runs in all most linear time with respect to the size of trees, while the computation time of \mathbf{K}_t increases drastically.

8.2.2 Glycan Data Classification by Spectrum Tree Kernel

In this experiment, we compare our spectrum tree kernel with the labeled ordered tree kernel \mathbf{K}_L in supervised classification problems. Concretely, we deal with glycan structure classification problems in the field of bioinformatics.

We had fourteen kinds of node labels. We have summarized the data used in our experiments in Table 8.1.

We used LIBSVM [CL01] as the SVM implementation, and used the area under the ROC curve (AUC) as the performance measure. The AUC is a prevailing performance measure of a decision function with a kernel to separate positive examples from negative ones. The AUC values range from 0.5 to 1.0, where the value 0.5 indicates a random separation whereas the value 1.0 indicates a perfect separation.

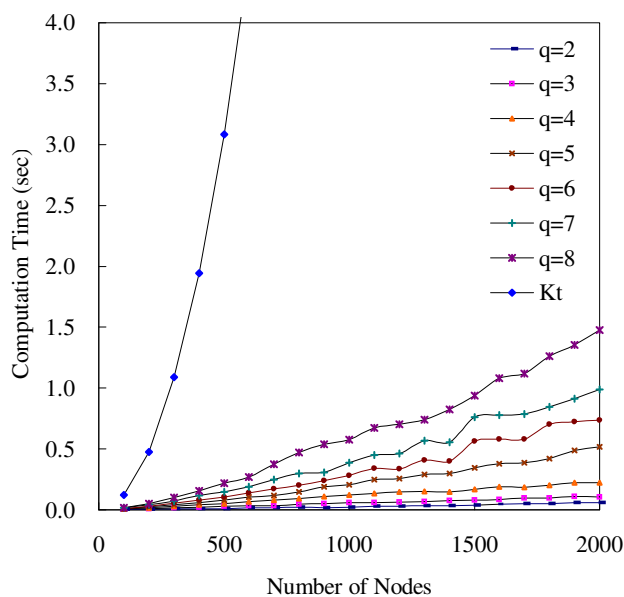


Figure 8.1. The running time for computing K_t and K_q

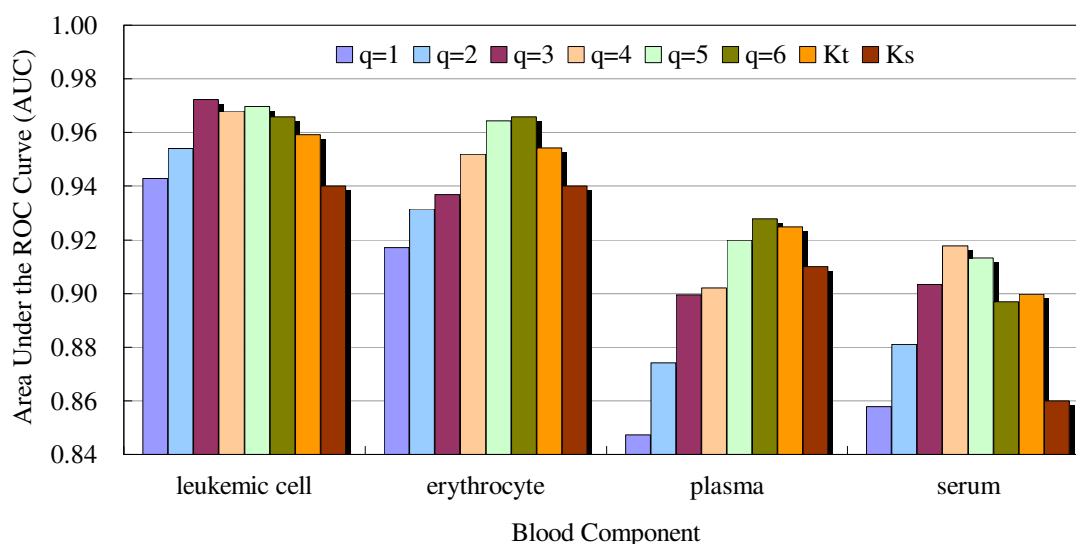


Figure 8.2. Area under the ROC curve

8.2.3 Predictive Accuracy

Figure 8.2 shows the comparison of the results by the proposed method with varying the parameter q , the labeled ordered tree kernels K_t [KK02], and the tree kernel due to Vishwanathan and Smola K_s [VS02] (in the area under the ROC curve (AUC)). All the performance measures were measured by 5-fold cross validation.

The approach by Hizukuri *et al.* [HYN⁺05] roughly corresponds to the case $q = 3$ (with various biological heuristics), but the spectrum tree kernel achieves the better performances at larger q except for the class *leukemic cell*. This result supports effectiveness of incorporating various structural contexts in trees.

The tree kernel due to Vishwanathan and Smola also gave relatively good performances in spite of its restricted expressive power. Since the nodes near the leaves tend to determine the functionalities of glycans, this data set seems to be well-suited to this tree kernel.

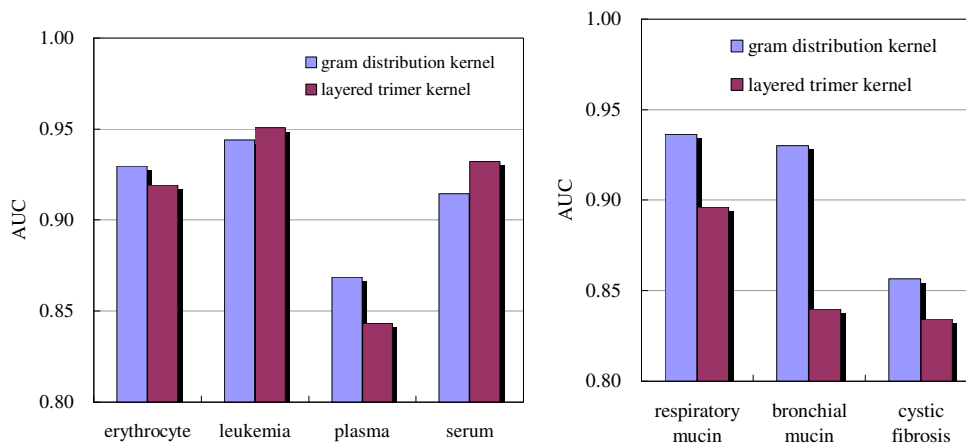


Figure 8.3. The performances of the SVM classifier for the gram distribution kernel and the layered trimer kernel

It is surprising that the spectrum tree kernel outperforms the labeled ordered tree kernels in spite of its expressiveness of structured information, which indicates that the expressive power of the spectrum tree kernel is moderate for glycan data, and prevents overfitting to the training data. This encourages us to apply our kernel to the data in other application domains on which the spectrum tree kernel performs better, but still with almost linear time (quasi-linear time) for kernel computation.

Also, it is interesting to point out that the value q achieving the predictive performance varies among the class labels, which indicates that the effective length of the patterns varies among class labels.

8.2.4 Motif Extraction by Gram Distribution Kernel

We use the decision value $\delta(x)$ obtained from the trained SVM to evaluate the contribution of each feature (i.e. q -gram pattern) in order to identify the glycan substructures characteristic to the target class. We define the *feature score* $F(f)$ [HYN⁺05] to indicate the significance of a feature f to be

$$F(f) = \sum_{x \in X} \delta(x) \cdot I_x(f),$$

where $I_x(f)$ is the indicator function defined by $I_x(f) = 1$ if x contains a feature f .

Features with large absolute values of feature scores are indicate *motifs* of glycan substructures playing key roles in discriminating the class label. Furthermore, we can compose larger and more complex substructures by overlapping more than one q -gram pattern.

8.2.5 Results and Discussion

We have evaluate the effectiveness of our gram distribution kernel by empirically comparing its predictive performance against the layered trimer kernel on glycan data. **Figure 8.3** illustrates the performance of the SVM classifier for the two experiments as stated in the following subsections. For classifying multi-classes, we employ the one-vs-rest approach with a binary classifier SVM. Performance is compared using the area under the ROC curve (AUC) measured by 5-fold cross validation.

Leukemia-Specific Features

As shown in the left graph in Figure 8.3, our kernel and the layered trimer kernel show almost identical performance. Since the top-scoring features are 3-mers as shown in the left graph in **Figure 8.4**, the layered trimer kernel is well fit to these data sets.

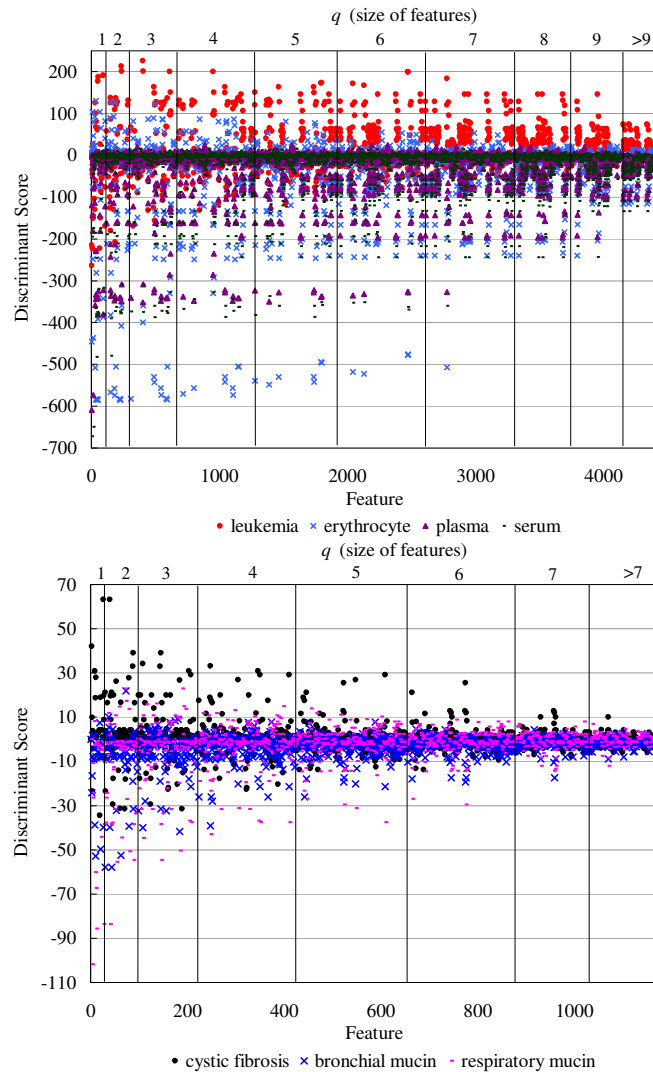


Figure 8.4. *The distributions of feature scores*

As listed in Table 8.2, the top-scoring features in the leukemia data set are in the 200 range and completely consists of subsets, or completely match the top-scoring substructure from the results by the layered trimer kernel. The second top scoring structure also comes immediately after this group of substructures, scoring over 200. Correspondingly, on the erythrocyte data set, the top scores are around 120 and consists completely of substructures, or completely match the high scoring substructures from the layered trimer kernel. Note that in our experiments, we incorporate information indicating root and leaf nodes, which we indicate in this table.

In contrast, we have obtained different resulting features for the serum and plasma data sets. However, this can be explained by the fact that these data sets were less specific, resulting in the low scores from both methods. Thus, although our method does not require any special weighting techniques, it has been able to produce similar results as the previous method. On top of that our method allows us to find substructures larger than trimer structures. In fact, a 6-mer structure scoring 200 is also obtained.

Features Captured for Cystic Fibrosis

We first look at the accuracy performance of our method and compare it to the layered trimer method. We find that the AUC scores are much higher for all three data sets, compare to the slightly higher performance of this previous method on the leukemia data set (See Figure 8.3).

Table 8.2. Features extracted by our method

leukemia	
Score	Substructure
226	(leaf)NeuAc2- α 3Gal- β 4
201	(leaf)NeuAc2- α 6Gal- β 4
201	(leaf)NeuAc2- α 3Gal- β 4GlcNAc- β 2
200	-Gal- β 4GlcNAc- β 2Man- α 6Man- β 4GlcNAc- β 4GlcNAc(root)
200	-Gal- β 4GlcNAc- β 2Man- α 3Man- β 4GlcNAc- β 4GlcNAc(root)
erythrocyte	
Score	Substructure
122	-Gal- β 4GlcNAc- β 3Gal- β 4
86	(leaf)Fuc- α 2Gal- β 4GlcNAc- β 3
86	-GlcNAc- β 3Gal- β 4Glc- β 1(root)
82	-Gal- β 4GlcNAc- β 3Gal- β 4GlcNAc- β 3
81	(leaf)Fuc- α 2Gal- β 4GlcNAc- β 3Gal- β 4

Note that 1-mer and 2-mer substructures are found to have the top scores as shown in the right graph in Figure 8.4. Looking at the features extracted from our method, the top scoring CF-related structures scored 63 and represent α 2 - 3 sialylated structures, which corresponds with the literature [MLGB92, DDRL99]. The second top scoring structure scored 39 and represents the sialylated galactose which are often found at the non-reducing ends of these structures. We also find the 6-sulfated GlcNAcs in the higher scoring range, as also mentioned in the literature [DDRL99]. In contrast, the highest scoring structures from the bronchial and respiratory data sets are the non-sialylated substructures of the O-glycan core. This further supports the possibility of the sialylated galactose substructure as being characteristic of CF.

8.3 Summary

In this work, we focus on explicit feature extraction from tree structured data. In order to assess the performance of our new method, comparable kernels included the layered trimer kernel [HYN⁺05] and Vishwanathan and Smola [VS02]. The latter is not included in our experiments due to its limited expressive power; it only considers entire subtrees and cannot extract the internal structures as features. Other kernels considered are the Collins and Duffy kernel [CD01] and the Kashima and Koyanagi kernel [KK02]. However, both of these methods implicitly enumerate features; therefore, they cannot be used to directly extract motifs from our data sets. Thus, the only kernel available that can be applied directly to glycans is the trimer kernel, and our method outperforms it in the experiments. In addition, our method can be used for other glycan data sets.



Chapter 9

Conclusion and Future Work

This chapter summarizes the results of this work, and concludes by discussing further issues to be addressed.

9.1 Conclusion

This thesis has focused mainly on two problems relating to tree structured data. Firstly, we have addressed the tree-to-tree comparison problem based on edit distance. Secondly, we have applied the resulting theory established in the first problem to a tree classification problem based on kernel methods, and developed novel learning methods.

The notion of *tree mappings* allows us to have a uniform approach to two different problems, the matching and learning problems in trees. These two problems are regarded as the following combinatorial problems.

Edit-based tree matching: An *optimization problem* of tree mappings, in which a minimum cost of tree mappings between two trees gives a common tree pattern or a distance measure.

Kernel-based tree learning: A *counting problem* of tree mappings, in which the number of tree mappings between two trees gives a similarity or kernel function for learning trees.

In what follows, we review the more specific results of these two subjects and our contributions along with the outline of this study. We began this thesis with a review of prior work on approximate tree matching in Chapter 2, and gave the strict definitions of existing methods based on partially ordered set theory instead of conventional ones. A variety of tree edit distance measures have been proposed in the past three decades such as Tai distance, alignment distance, less-constrained distance, constrained distance, structure-preserving distance, structure-respecting distance, top-down distance, bottom-up distance, and so forth. These measures were described mainly in two ways, i.e. operational description and declarative description. An operational definition of a tree edit distance measure describes *how* to compute the distance by showing the procedure or the algorithm, whereas an declarative definition of a measure describes *what* the measure is by means of a set-theoretical treatment. In particular, the notion of tree mapping has been used in the declarative definition since Tai showed his distance measure is defined by using tree mapping. A tree mapping depicts node-to-node correspondences between two trees according to the structural similarity. During this review, we have identified a number of confusions and unsolved problems in prior work on tree edit distance. These problems include the following:

- The declarative definitions of less-constrained distance given by Lu *et al.* is incorrect. It does not coincide with the algorithm, and it defines a different distance from what they originally intended;
- The tree mapping of alignment distance has been unknown for the past decade, i.e. there has not been a declarative definition of alignment distance;
- Equivalent distance measures have been repeatedly proposed without being recognized.

These problems were all caused by the lack of a theoretical foundation for describing the semantics of tree edit distance, and a means of bridging the gap between operational definitions and declarative definitions. To surmount these problems, we have constructed a mathematical model of tree edit distance by using partially ordered set theory in Chapter 3. This theoretical foundation enables us to deal with the semantics of tree edit distance in a rigorous way. In Chapter 4, we have revealed the following facts:

- We showed a correct declarative definition of less-constrained distance, and that the definition given by Lu *et al.* turned out to represent the constrained distance given by Zhang.
- We identified the condition of the tree mapping of alignment distance, i.e. a declarative definition of alignment distance.
- We proved the equivalence among the strongly structure-preserving distance, structure-respecting distance, and constrained distance. We also proved the equivalence between the alignment distance and less-constrained distance.

Furthermore, we showed a hierarchical relationship among these edit distance measures.

The last half part of this thesis deals with a tree classification problem with the Support Vector Machine (SVM) based on kernel-based learning. In particular, we have focused on a kernel design problem for trees. In Chapter 5, we gave a cursory review of tree kernels presented in prior work. From this review, we found that some of these tree kernels are characterized by some classes of tree mappings. In fact, we showed that a tree kernel proposed by Kashima and Koyanagi is the counting function of tree mappings in a class, i.e. the accordant mapping. From this observation, we extrapolated that counting functions for the other classes of tree mapping would also form new tree kernels. Then we applied the theoretical foundation that we developed in the first half of this thesis to the design problem of tree kernels in Chapter 6. Specifically, we proposed the algorithms for computing counting functions of Tai mappings, alignable mappings, and semi-accordant mappings. We then showed that the counting functions of Tai mappings and semi-accordant mappings are actually tree kernels and that these two tree kernels have more flexible expressive power than the tree kernel proposed by Kashima and Koyanagi. In contrast, we showed that the counting function of alignable mappings is not a tree kernel. All of these results have confirmed and proved the effectiveness of our theoretical foundation of approximate tree matching.

In the next chapter, we aimed to develop a faster tree kernel without sacrificing its learning performance as compared with the tree kernels proposed by Kashima and Koyanagi, which runs in quadratic time with respect to the size of trees. Then we proposed a spectrum tree kernel based on the notion of tree q -gram, which runs in almost linear time. In addition, we proposed its variant, a gram distribution kernel. The basic idea of the spectrum tree kernel is that the more the same subpatterns are shared in two trees, the more similar these trees are.

Finally, in Chapter 8, we evaluated the effectiveness of the two kernel based on tree q -gram by empirically comparing its computation time and predictive performance in a glycan structure classification problem with the times and performances of existing methods. We attained a good performance with our tree kernels although we do not incorporate any biological knowledge specific to glycan data classification. Moreover, by using the trained SVM, we successfully extracted common characteristic substructures specific to a class of glycans.

9.2 Future Work

In the first half part of this thesis, we focused on the tree-to-tree comparison problem between two labeled rooted trees. Two important problems immediately emerge from our problem setting by extension.

Firstly, we considered only pairwise comparison of trees in this thesis. A variety of multiple tree comparison problems can be considered by extension such as a common sub/super-tree pattern problem shared in more than two tree. From the theoretical point of view, it is intriguing to extend the notion of each class of tree mapping to more than two trees, and to investigate the property of tree mapping. From a practical point of view, if a tree mapping among multiple trees could be efficiently computed, it would provide a general method for common pattern discovery in trees, and a wide range of applications would be expected including motif extraction from biological data, schema extraction from XML data, and so forth.

Secondly, another challenging problem still to be addressed is the extension of our algebraic formulation for trees to more general graph structures such as directed acyclic graphs, and planar lattices, in order to develop new approximate matching methods for these structures. There is still need for more fundamental investigation of this area since a widely-accepted model of the edit distance for general graphs has yet to be established.

In the second part of this thesis, we developed several tree kernels with high expressiveness. The expressiveness is determined by the pattern language by which the number of subpattern occurrences in trees are counted in computing the value of a kernel function. In this work, we employed several classes of tree mappings and q -gram as pattern languages for tree kernels. However, the trade-off between two factors of expressiveness of the pattern language and the learning performance has not been well investigated. In fact, we expect that some pattern languages with high expressiveness such as Tai mapping and semi-accordant mapping may not necessarily lead to a sufficient learning performance, since Bringmann *et al.* reported in [BZRN06] that the use of more complex patterns does not necessarily lead to better accuracy. Identification of the trade-offs between our pattern languages is therefore an important area for further investigation.

We reviewed some tree kernels which were originally thought to be in the class of Hausler's convolution kernel. To our surprise, the elastic tree kernel proposed by Kashima and Koyanagi turned out to be beyond this class. Nevertheless, we did show that this class of counting functions is also guaranteed to be a kernel. This fact implies that our mapping kernel leads to a more general framework superseding the convolution kernel for designing kernels of discrete structures.

Finally, from a practical point of view, the tree kernels proposed in this thesis should be applied to a wider variety of real-world tree structured data other than glycan data including XML documents and RNA secondary structures.

Closing Remarks

Much work remains to be done. I, however, believe that the findings of this thesis provide fundamental and effective contributions to assist in solving problems related to tree-to-tree comparison. I hope that these results will encourage further advances in our understanding of approximate tree matching, and lead to deeper insights into related problems.

Bibliography

- [AAK⁺02] T. Asai, K. Abe, S. Kawazoe, H. Arimura, H. Sakamoto, and S. Arikawa, *Efficient substructure discovery from large semi-structured data*, Proc. of 2nd SIAM International Conference on Data Mining (SDM), 2002.
- [ABG05] N. Augsten, M. H. Böhlen, and J. Gamper, *Approximate matching of hierarchical data using pq-grams.*, Proc. of 31st International Conference on Very Large Data Bases (VLDB), 2005, pp. 301–312.
- [ACG⁺02] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, and A. Marchetti-Spaccamela, *Complexity and approximation - combinatorial optimization problems and their approximability properties*, 2nd ed., Springer-Verlag, 2002.
- [AFT06] T. Akutsu, D. Fukagawa, and A. Takasu, *Approximating tree edit distance through string edit distance*, Proc. of Third International Conference on Distributed Computing and Internet Technology (ICDCIT), ISSAC, Lecture Notes in Computer Science, 2006, pp. 90–99.
- [AH00] T. Akutsu and M. M. Halldórsson, *On the approximation of largest common subtrees and largest common point sets*, Theoretical Computer Science **233** (2000), 33–50.
- [Aku92] T. Akutsu, *An RNC algorithm for finding a largest common subtree of two trees*, IEICE Trans. Information and Systems **E75-D** (1992), 95–101.
- [Aku96] T. Akutsu, *Approximate string matching with variable length don't care characters*, IEICE Transactions on Communications/Electronics/Information and Systems **E78-D** (1996), no. 9, 1353–1354.
- [Aku00] T. Akutsu, *Dynamic programming algorithms for RNA secondary structure prediction with pseudoknots*, Discrete Applied Mathematics **104** (2000), 45–62.
- [Aku06] T. Akutsu, *A relation between edit distance for ordered trees and edit distance for euler strings*, Information Processing Letter **100** (2006), no. 3, 105–109.
- [AKUMK06] K. F. Aoki-Kinoshita, N. Ueda, H. Mamitsuka, and M. Kanehisa, *ProfilePSTMM: capturing tree-structure motifs in carbohydrate sugar chains*, ISMB, 2006, pp. 25–34.
- [Aok83] K. Aoki, *An algorithm computing the distance of tai between two labeled ordered trees*, Transactions of the Institute of Electronics and Communication Engineers of Japan **J66-D** (1983), no. 1, 49–56, (In Japanese).
- [AS04] J. Allali and M.-F. Sagot, *Novel tree edit operations for RNA secondary structure comparison*, WABI 2004, LNBI 3240, 2004, pp. 412–425.
- [AYO⁺03] K. F. Aoki, A. Yamaguchi, Y. Okuno, T. Akutsu, N. Ueda, M. Kanehisa, and H. Mamitsuka, *Efficient tree-matching methods for accurate carbohydrate database queries*, Genome Informatics **14** (2003), 134–143.
- [Bad06] M. Badoiu, *Algorithmic embeddings*, Ph.D. thesis, Massachusetts Institute of Technology, 2006.
- [BBP04] S. Berretti, A. Del Bimbo, and P. Pala, *A graph edit distance based on node merging*, Proc. of Image and Video Retrieval: Third International Conference (CIVR), Lecture Notes in Computer Science, vol. 3115, 2004, pp. 464–472.

- [BCD95] D. Barnard, G. Clarke, and N. Duncan, *Tree-to-tree correction for document trees*, Tech. Report 95-375, Queen's University, Kingston, Ontario K7L 3N6 Canada, January 1995.
- [BDFW95] H. L. Bodlaender, R. G. Downey, M. R. Fellows, and H. T. Wareham, *The parameterized complexity of sequence alignment and consensus*, Theoretical Computer Science **147** (1995), no. 1–2, 31–54.
- [BES06] T. Batu, F. Ergün, and S. C. Sahinalp, *Oblivious string embeddings and edit distance approximations*, Proc. of 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2006, pp. 792–801.
- [BG05] P. Bille and I. L. Gørtz, *The tree inclusion problem: In optimal space and faster*, Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP), Lecture Notes in Computer Science, vol. 3580, 2005, pp. 66–77.
- [BHR00] L. Bergroth, H. Hakonen, and T. Raita, *A survey of longest common subsequence algorithms*, Proc. of 17th International Symposium on String Processing Information Retrieval (SPIRE), 2000, pp. 39–48.
- [Bil03] P. Bille, *Ordered tree edit distance with merge and split operations*, IT University Technical Report Series TR-2003-35, IT University of Copenhagen, September 2003.
- [Bil05] P. Bille, *A survey on tree edit distance and related problems*, Theoretical Computer Science **337** (2005), no. 1–3, 217–239.
- [BJK00] H. Bunke, X. Jiang, and A. Kandel, *On the minimum common supergraph of two graphs*, Computing **65** (2000), 13–25.
- [BK03] S. Burkhardt and J. Kärkkäinen, *Better filtering with gapped q-grams*, Fundamenta Informaticae **56** (2003), no. 1–2, 51–70.
- [BPS00] I. M. Bomze, M. Pelillo, and V. Stix, *Approximating the maximum weight clique using replicator dynamics*, IEEE Transaction on Neural Networks **11** (2000), no. 6, 1228–1241.
- [BS98] H. Bunke and K. Shearer, *A graph distance metric based on the maximal common subgraph*, Pattern Recognition Letters **19** (1998), 255–259.
- [Bun97] H. Bunke, *On a relation between graph edit distance and maximum common subgraph*, Pattern Recognition Letters **18** (1997), 689–694.
- [BZRN06] B. Bringmann, A. Zimmermann, L. De Raedt, and S. Nijssen, *Don't be afraid of simpler patterns*, Proc. of 10th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), Lecture Notes in Computer Science, vol. 4213, 2006, pp. 55–66.
- [CD01] M. Collins and N. Duffy, *Convolution kernels for natural language*, Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001], MIT Press, 2001, pp. 625–632.
- [CGM97] S. S. Chawathe and H. Garcia-Molina, *Meaningful change detection in structured data*, Proc. of the ACM SIGMOD International Conference on Management of Data, 1997, pp. 26–37.
- [Cha99a] S. S. Chawathe, *Comparing hierarchical data in external memory*, Proc. of the 25th International Conference on Very Large Data Bases (Edinburgh, Scotland, U.K.), 1999, pp. 90–101.
- [Cha99b] S.S. Chawathe, *Managing change in heterogeneous autonomous databases*, Ph.D. thesis, Stanford University, 1999.
- [Che98] Weimin Chen, *More efficient algorithm for ordered tree inclusion*, Journal on Algorithms **26** (1998), no. 2, 370–385.
- [Che01] W. Chen, *New algorithm for ordered tree-to-tree correction problem*, Journal of Algorithm **40** (2001), 135–158.

- [CL01] C.-C. Chang and C.-J. Lin, *LIBSVM: a library for support vector machines*, 2001, Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>.
- [CLRS01] T. H. Cormen, C. E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to algorithms*, MIT Electrical Engineering and Computer Science, The MIT Press, 2001.
- [CLZU02] M. Crochemore, G. M. Landau, and M. Ziv-Ukelson, *A sub-quadratic sequence alignment algorithm for unrestricted cost matrices*, Proc. of 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2002, pp. 679–688.
- [CM02] G. Cormode and S. Muthukrishnan, *The string edit distance matching problem with moves*, SODA, 2002, pp. 667–676.
- [CM07] G. Cormode and S. Muthukrishnan, *The string edit distance matching problem with moves*, ACM Transactions on Algorithms **3** (2007), no. 1.
- [Cor03] G. Cormode, *Sequence distance embeddings*, Ph.D. thesis, University of Warwick, January 2003.
- [CR02] M. Crochemore and W. Rytter, *Jewels of stringology — text algorithms*, World Scientific Publishing, Hong-Kong, 2002, 310 pages.
- [CRGMW96] S. S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom, *Change detection in hierarchically structured information*, Proceedings of the ACM SIGMOD International Conference on Management of Data, 1996, pp. 493–504.
- [CS04] A. Culotta and J. S. Sorensen, *Dependency tree kernels for relation extraction*, Proc. of 42nd Annual Meeting of the Association for Computational Linguistics (ACL), 2004, pp. 423–429.
- [DA92] S. Doubet and P. Albersheim, *Carbbank*, Glycobiology **2** (1992), no. 6.
- [DDRL99] S. Degroote, M. P. Ducourouble, P. Roussel, and G. Lamblin, *Sequential biosynthesis of sulfated and/or sialylated Lewis x determinants by transferases of the human bronchial mucosa*, Glycobiology **9** (1999), no. 11.
- [DMRW07] E. Demaine, S. Mozes, B. Rossman, and O. Weimann, *An optimal decomposition algorithm for tree edit distance*, Proc. of the 34th International Colloquium on Automata, Languages and Programming (ICALP), 2007.
- [DST80] P. J. Downey, R. Sethi, and R. E. Tarjan, *Variations on the common subexpression problem*, Journal of the Association for Computing Machinery (J. ACM) **27** (1980), no. 4, 758–771.
- [DT03a] S. Dulucq and L. Tichit, *Rna secondary structure comparison: exact analysis of the zhang-shasha tree edit algorithm*, Theoretical Computer Science **306** (2003), no. 1–3, 471–484.
- [DT03b] S. Dulucq and H. Touzet, *Analysis of tree edit distance algorithms*, Proc. in 14th Annual Symposium on Combinatorial Pattern Matching (CPM), Lecture Notes of Computer Science, vol. 2676, 2003, pp. 83–95.
- [DT05] S. Dulucq and H. Touzet, *Decomposition algorithms for the tree edit distance problem*, Journal of Discrete Algorithms **3** (2005), no. 2–4, 448–471.
- [FA06] Daiji Fukagawa and Tatsuya Akutsu, *Fast algorithms for comparison of similar unordered trees*, International Journal of Foundations of Computer Science **17** (2006), no. 3, 703–729.
- [FG00] P. Ferraro and C. Godin, *A distance measure between plant architectures*, Annals of Forest Science **57** (2000), 445–461.
- [FG03] P. Ferraro and C. Godin, *An edit distance between quotiented trees*, Algorithmica **36** (2003), 1–39.
- [FO05] P. Ferraro and A. Ouangraoua, *Local mapping between unordered trees*, Tech. Report RR-105, LaBRI, 2005.

- [FRV04] M. V. Ferro, F. J. Ribadas, and J. Vilares, *Phrase similarity through the edit distance*, Proc. of Database and Expert Systems Applications, 15th International Conference, DEXA 2004, Lecture Notes in Computer Science, vol. 3180, 2004, pp. 306–317.
- [FSS90] P. Flajolet, P. Sipala, and J.-M. Steyaert, *Analytic variations on the common subexpression problem*, Automata, Languages and Programming, Lecture Notes in Computer Science, vol. 443, Springer Verlag, 1990, pp. 220–234.
- [GB02] S. Günter and H. Bunke, *Self-organizing map for clustering in the graph domain*, Pattern Recognition Letters **23** (2002), 405–417.
- [GJ79] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman And Company, New York, 1979.
- [GK05] M. Garofalakis and A. Kumar, *XML stream processing using tree-edit distance embeddings*, ACM Transactions on Database Systems **30** (2005), no. 1, 279–332.
- [Gus97] D. Gusfield, *Algorithms on strings, trees, and sequences: Computer science and computational biology*, Cambridge University Press, 1997.
- [HAKU⁺06] K. Hashimoto, K. F. Aoki-Kinoshita, N. Ueda, M. Kanehisa, and H. Mamitsuka, *A new efficient probabilistic model for mining labeled ordered trees*, KDD, 2006.
- [Hau99] D. Haussler, *Convolution kernels on discrete structures*, UCSC-CRL 99-10, Dept. of Computer Science, University of California at Santa Cruz, 1999.
- [HKG⁺06] K. Hashimoto, S. Goto, S. Kawano, K. F. Aoki-Kinoshita, and N. Ueda, *Kegg as a glycome informatics resource*, Glycobiology **16** (2006), 63R–70R.
- [HK05] A. Hogue and D. Karger, *Thresher: Automating the unwrapping of semantic content from the world wide web*, Proc. of 14th International World Wide Web Conference (WWW), 2005, pp. 86–95.
- [HM76] J. W. Hunt and M. D. McIlroy, *An algorithm for differential file comparison*, Tech. Report CSTR #41, Bell Telephone Laboratories, 1976.
- [HMU06] Y. Horsesh, R. Mehr, and R. Unger, *Designing an a* algorithm for calculating edit distance between rooted-unordered trees*, Journal of Computational Biology **13** (2006), no. 6, 1165–1176.
- [Höc05] M. Höchsmann, *The tree alignment model: Algorithms, implementations and applications for the analysis of rna secondary structures*, Ph.D. thesis, Technishhen Fakultät der Universität Bielefeld, 2005.
- [HTGK03] M. Höchsmann, T. Töller, R. Giegerich, and S. Kurtz, *Local similarity in RNA secondary structures*, Proc. of the Computational Systems Bioinformatics (CSB), IEEE, 2003, pp. 159–168.
- [HYHK04] Y. Hizukuri, Y. Yamanishi, K. Hashimoto, and M. Kanehisa, *Extraction of species-specific glycan substructures*, Genome Informatics. Proc. of International Conference on Genome Informatics (GIW), vol. 1, 2004, pp. 69–81.
- [HYN⁺05] Y. Hizukuri, Y. Yamanishi, O. Nakamura, F. Yagi, S. Goto, and M. Kanehisa, *Extraction of leukemia specific glycan motifs in humans by computational glycomics*, Carbohydrate research **340** (2005), no. 14, 2270–2278.
- [Jan03] J. Jansson, *Consensus algorithms for trees and strings*, Ph.D. thesis, Lund University, Sweden, 2003.
- [JL03] J. Jansson and A. Lingas, *A fast algorithm for optimal alignment between similar ordered trees*, Fundamenta Informaticae **56** (2003), 105–120.

- [JU91] P. Jokinen and E. Ukkonen, *Two algorithms for approximate string matching in static texts*, Proc. of 16th Symposium on Mathematical Foundations of Computer Science (MFCS'91) LNCA, vol. 520, 1991, pp. 240–248.
- [JWZ95] T. Jiang, L. Wang, and K. Zhang, *Alignment of trees — an alternative to tree edit*, Theoretical Computer Science **143** (1995), 137–148.
- [Kan91] V. Kann, *Maximum bounded 3-dimensional matching in max snp-complete*, Information Processing Letter (IPL) **37** (1991), no. 1, 27–35.
- [KHAK⁺06] T. Kuboyama, K. Hirata, K. F. Aoki-Kinoshita, H. Kashima, and H. Yasuda, *A gram distribution kernel applied to glycan classification and motif extraction*, Genome Informatics **17** (2006), no. 2, 25–34.
- [KHK⁺06] T. Kuboyama, K. Hirata, H. Kashima, K. F. Aoki-Kinoshita, and H. Yasuda, *A spectrum tree kernel*, Proc. The International Workshop on Data Mining and Statistical Science (DMSS), 2006, pp. 109–116.
- [KHK⁺07] T. Kuboyama, K. Hirata, H. Kashima, K. F. Aoki-Kinoshita, and H. Yasuda, *A spectrum tree kernel*, Transactions of the Japanese Society for Artificial Intelligence **22** (2007), no. 2, 140–147.
- [KHOH06] T. Kuboyama, K. Hirata, N. Ohkura, and M. Harao, *A q-grams based distance for ordered labeled trees*, Proc. of 4th Workshop on Learning with Logics and Logics for Learning (LLLL), 2006, pp. 77–83.
- [KK02] H. Kashima and T. Koyanagi, *Kernels for semi-structured data*, Proc. of 9th International Conference on Machine Learning (ICML), 2002, pp. 291–298.
- [KKSS03] K. Kailing, H.-P. Kriegel, S. Schönauer, and T. Seidl, *Efficient similarity search for hierarchical data in large databases*, Proc. of 9th International Conference on Extending Database Technology (EDBT), Lecture Notes in Computer Science, vol. 2992, 2003.
- [Kle98] P. N. Klein, *Computing the edit-distance between unrooted ordered trees*, Proc. of 6th Annual European Symposium on Algorithms (ESA), Lecture Notes in Computer Science, vol. 1461, 1998, pp. 91–102.
- [KLM⁺00] J. D. Kececioğlu, H.-P. Lenhof, K. Mehlhorn, P. Mutzel, K. Reinert, and M. Vingron, *A polyhedral approach to sequence alignment problems*, Discrete Applied Mathematics **104** (2000), 143–186.
- [KM95] P. Kilpeläinen and H. Mannila, *Ordered and unordered tree inclusion*, SIAM Journal on Computing **24** (1995), no. 2, 340–356.
- [KR06] R. Krauthgamer and Y. Rabani, *Improved lower bounds for embeddings into l_1* , Proc. of 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2006, pp. 1010–1017.
- [KS01] T. Kahveci and A. K. Singh, *Efficient index structures for string databases*, Proc. of 27th International Conference on Very Large Data Bases (VLDB), 2001, pp. 351–360.
- [KSK06a] H. Kashima, H. Sakamoto, and T. Koyanagi, *Tree kernels*, Transactions of the Japanese Society for Artificial Intelligence **21** (2006), no. 1, 113–121, (In Japanese).
- [KSK06b] T. Kuboyama, K. Shin, and H. Kashima, *Flexible tree kernels based on counting the number of tree mappings*, Proc. of 4th International Workshop on Mining and Learning with Graphs (MLG) in conjunction with ECML/PKDD, 2006, pp. 61–72.
- [KSM05] T. Kuboyama, K. Shin, and T. Miyahara, *A theoretical analysis of tree edit distance measures*, Transactions on Mathematical Modeling and its Applications (TOM13), The Information Processing Society of Japan **46** (2005), no. 17, 31–45.
- [KSMY05] T. Kuboyama, K. Shin, T. Miyahara, and H. Yasuda, *A theoretical analysis of alignment and edit problems for trees*, Proc. of 9th Italian Conference of Theoretical Computer Science (ICTCS), Lecture Notes in Computer Science, vol. 3701, 2005, pp. 323–337.

- [KWU06] J. B. Kruskal, M. Wish, and E. M. Uslander, *Multidimensional scaling (quantitative applications in the social sciences)*, Sage Publications, 2006.
- [LBLL⁺06] T. Lutteke, A. Bohne-Lang, A. Loss, T. Goetz, M. Frank, and C.W. von der Lieth, *GLYCO-SCIENCES.de: an internet portal to support glycomics and glycobiology research*, *Glycobiology* **16** (2006), no. 5.
- [LEN02] C. S. Leslie, E. Eskin, and W. S. Noble, *The spectrum kernel: A string kernel for svm protein classification*, Pacific Symposium on Biocomputing, vol. 7, 2002, pp. 566–575.
- [Lev66] V. I. Levenshtein, *Binary codes capable of correcting insertions and reversals*, Soviet Physics Doklady **10** (1966), no. 8, 707–710.
- [LSST⁺02] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. J. C. H. Watkins, *Text classification using string kernels.*, *Journal of Machine Learning Research* **2** (2002), 419–444.
- [LST01] C. L. Lu, Z.-Y. Su, and C. Y. Tang, *A new measure of edit distance between labeled trees*, COCOON, Lecture Notes in Computer Science, vol. 2108, 2001, pp. 338–348.
- [Lu79] S.-Y. Lu, *A tree-to-tree distance and its application to cluster analysis*, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **1** (1979), no. 2, 219–224.
- [Mat78] D. W. Matula, *Subtree isomorphism in $O(n^{5/2})$* , *Annals of Discrete Mathematics* **2** (1978), 91–106.
- [MLGB92] T. P. Mawhinney, D. C. Landrum, D. A. Gayer, and G. J. Barbero, *Sulfated sialyl-oligosaccharides derived from tracheobronchial mucous glycoproteins of a patient suffering from cystic fibrosis*, *Carbohydr. Res.* **235** (1992).
- [Mos06] Alessandro Moschitti, *Efficient convolution kernels for dependency and constituent syntactic trees*, Proc. of 17th European Conference on Machine Learning, Berlin (ECML), Lecture Notes in Computer Science, vol. 4212, 2006, pp. 318–329.
- [MP80] W. J. Masek and M. Paterson, *A faster algorithm computing string edit distances.*, *Journal of Computer and System Sciences* **20** (1980), no. 1, 18–31.
- [MR07] F. Magniez and M. de Rougemont, *Property testing of regular tree languages*, *Algorithmica* (2007), To appear, (<http://www.lri.fr/~magniez/PAPIERS/mr-algorithmica06.pdf>).
- [MS05] Dinesh P. Mehta and Sartaj Sahni (eds.), *Handbook of data structures and applications*, Chapman & Hall/CRC Computer & Information Science Series, CRC Press, 2005.
- [MT92] J. Matoušek and R. Thomas, *On the complexity of finding iso- and other morphisms for partial k -trees*, *Discrete Mathematics* **108** (1992), no. 1–3, 343–364.
- [MTL02] B. Ma, J. Tromp, and M. Li, *PatternHunter: Faster and more sensitive homology search*, *Bioinformatics* **18** (2002), 440–445.
- [Mye99] G. Myers, *A fast bit-vector algorithm for approximate string matching based on dynamic programming*, *Journal of the Association for Computing Machinery (J. ACM)* **46** (1999), no. 3, 395–415.
- [NBYST01] G. Navarro, R. Baeza-Yates, E. Sutinen, and J. Tarhio, *Indexing methods for approximate string matching*, *IEEE Data Engineering Bulletin* **24** (2001), no. 4, 19–27.
- [NR02] G. Navarro and M. Raffinot, *Flexible pattern matching in strings — practical on-line search algorithms for texts and biological sequences*, Cambridge University Press, 2002, ISBN 0-521-81307-7. 280 pages.
- [NW70] S. B. Needleman and C. D. Wunsch, *A general method applicable to the search for similarities in the amino acid sequence of two proteins*, *Journal of Molecular Biology* **48** (1970), 43–453.

- [OHKH05] N. Ohkura, K. Hirata, T. Kuboyama, and M. Harao, *The q -gram distance for ordered unlabeled trees*, Proc. of 8th International Conference on Discovery Science (DS), Lecture Notes in Computer Science, vol. 3735, 2005, pp. 189–202.
- [Ols05] O. F. Olsen, *Tree edit distances from singularity theory*, Proc. in 5th International Conference on Scale Space and PDE Methods in Computer Vision, Lecture Notes in Computer Science, vol. 3459, 2005, pp. 316–326.
- [OR05] R. Ostrovsky and Y. Rabani, *Low distortion embeddings for edit distance*, Proc. of 37th Annual ACM Symposium on Theory of Computing (STOC), 2005, pp. 218–224.
- [OT88] K. Ohmori and E. Tanaka, *A unified view on tree metrics*, pp. 85–100, Springer-Verlag New York, Inc., 1988, Nato Asi Series.
- [PFR06] A. Passerini, P. Frasconi, and L. De Raedt, *Kernels on prolog proof trees: Statistical learning in the ILP setting*, Journal of Machine Learning Research **7** (2006), 307–342.
- [Rah07] S. Rahmann, *Foundation of sequence analysis*, Jan 2007, Lecture notes for a course in the Winter Semester 2006/07.
- [Ric97] T. Richter, *A new measure of the distance between ordered trees and its applications*, Tech. Report 85166-CS, Dept. of Computer Science, Univ. of Bonn, 1997.
- [RKH02] A. Robles-Kelly and E. R. Hancock, *String edit distance, random walks and graph matching*, Proc. in Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition, Lecture Notes in Computer Science, vol. 2396, 2002, pp. 104–112.
- [RKH03] A. Robles-Kelly and E. R. Hancock, *Graph matching using spectral seriation and string edit distance*, Proc. in 4th IAPR International Workshop on Graph Based Representations in Pattern Recognition (GbrPR), Lecture Notes in Computer Science, vol. 2726, 2003, pp. 154–165.
- [Sak03] Y. Sakakibara, *Pair hidden markov models on tree structures*, Bioinformatics **19** (2003), 232–240.
- [Sel77] S. M. Selkow, *The tree-to-tree editing problem*, Information Processing Letter (IPL) **6** (1977), no. 6, 184–186.
- [Sha48] C. E. Shannon, *A Mathematical Theory of Communication*, Bell System Techal Journal **27** (1948), 379–423, 623–656.
- [SI05] J. Suzuki and H. Isozaki, *Sequence and tree kernels with statistical feature mining*, Advances in Neural Information Processing Systems 18 [Neural Information Processing Systems (NIPS)], 2005.
- [SS02] D. Shapira and J. A. Storer, *Edit distance with move operations*, In Proc. of 13th Annual Symposium on Combinatorial Pattern Matching (CPM), Lecture Notes in Computer Science, vol. 2373, 2002, pp. 85–98.
- [STC04] J. Shawe-Taylor and N. Cristianini, *Kernel methods for pattern analysis*, Cambridge University Press, 2004.
- [SW81] T. F. Smith and M. S. Waterman, *Identification of common molecular subsequences*, Journal of Molecular Biology **147** (1981), 195–197.
- [SZ90] D. Shasha and K. Zhang, *Fast algorithms for the unit cost editing distance between trees*, Journal of Algorithms **11** (1990), 581–621.
- [SZ97] D. Shasha and K. Zhang, *Pattern matching algorithms*, ch. 14: Approximate Tree Pattern Matching, pp. 341–371, Oxford University Press, 1997.
- [Tai79] K.-C. Tai, *The tree-to-tree correction problem*, Journal of the Association for Computing Machinery (J. ACM) **26** (1979), no. 3, 422–433.

- [Tan83] E. Tanaka, *A bottom-up computing method for the metric between trees defined by tai*, Transactions of the Institute of Electronics and Communication Engineers of Japan **J66-D** (1983), no. 6, 660–667, (In Japanese).
- [Tan84] E. Tanaka, *The metric between trees based on the strongly structure preserving mapping and its computing method*, Transactions of the Institute of Electronics and Communication Engineers of Japan **J67-D** (1984), no. 6, 722–723, (In Japanese).
- [Tan93] E. Tanaka, *A note on a metric on trees and its computing method*, Transactions of the Institute of Electronics, Information and Communication Engineers of Japan **J76-D-I** (1993), no. 11, 635, (In Japanese).
- [Tan95] E. Tanaka, *A note on a tree-to-tree editing problem*, International Journal of Pattern Recognition and Artificial Intelligence **9** (1995), no. 1, 167–172.
- [TH02] A. Torsello and E. R. Hancock, *Matching and embedding through edit-union of trees*, Proc. in 7th European Conference on Computer Vision-Part III (ECCV), Lecture Notes in Computer Science, vol. 2352, 2002, pp. 822–836.
- [TH03a] A. Torsello and E. R. Hancock, *Computing approximate tree edit distance using relaxation labeling*, Pattern Recognition Letters **24** (2003), no. 8, 1089–1097.
- [TH03b] A. Torsello and E. R. Hancock, *Graph clustering with tree-unions*, Proc. of 10th International Conference of Computer Analysis of Images and Patterns (CAIP), Lecture Notes in Computer Science, vol. 2756, November 2003, pp. 451–459.
- [TH03c] A. Torsello and E. R. Hancock, *Learning mixtures of tree-unions by minimizing description length*, Proc. of 4th International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition (EMMCVPR), Lecture Notes in Computer Science, vol. 2683, 2003, pp. 130–146.
- [TH03d] A. Torsello and E. R. Hancock, *Learning structural variations in shock trees*, Proc. of Joint IAPR International Workshops on Structural, Syntactic, and Statistical Pattern Recognition, Lecture Notes in Computer Science, vol. 2396, 2003, pp. 113–122.
- [TH03e] A. Torsello and E. R. Hancock, *Tree edit distance from information theory*, Proc. of IAPR Workshop GbRPR, Lecture Notes in Computer Science, vol. 2726, 2003, pp. 71–82.
- [Tor04] A. Torsello, *Matching hierarchical structures for shape recognition*, Ph.D. thesis, Department of Computer Science, The University of York, 2004.
- [Tou03] H. Touzet, *Tree edit distance with gaps*, Information Processing Letters **85** (2003), 123–129.
- [Tou05] H. Touzet, *A linear tree edit distance algorithm for similar ordered trees*, Proc. of 16th Annual Symposium on Combinatorial Pattern Matching (CPM), Lecture Notes in Computer Science, vol. 3537, 2005, pp. 334–345.
- [TT82] E. Tanaka and K. Tanaka, *A metric on trees and its computing method*, Transactions of the Institute of Electronics and Communication Engineers of Japan **J65-D** (1982), no. 5, 511–518, (In Japanese), *errors in this article are corrected in [Tan93]*.
- [TT88] E. Tanaka and K. Tanaka, *The tree-to-tree editing problem*, International Journal of Pattern Recognition and Artificial Intelligence **2** (1988), no. 2, 221–240, *errors in this article are corrected in [Tan95]*.
- [Ukk92] E. Ukkonen, *Approximate string-matching with q-grams and maximal matches*, Theoretical Computer Science **92** (1992), no. 1, 191–211.
- [Ukk93] E. Ukkonen, *Approximate string-matching with q-grams and maximal matches*, Theoretical Computer Science **92** (1993), 191–211.
- [Val79] L. G. Valiant, *The complexity of computing the permanent*, Theoretical Computer Science **8** (1979), 189–201.

- [Val98] G. Valiente, *Algorithms on trees and graphs*, Springer, 1998.
- [Val01] G. Valiente, *An efficient bottom-up distance between trees*, Proc. of 8th International Symposium on String Processing and Information Retrieval (SPIRE), IEEE Computer Science Press, 2001, pp. 212–219.
- [Val05] G. Valiente, *Constrained tree inclusion*, Journal on Discrete Algorithms **3** (2005), no. 2–4, 431–447.
- [Vap95] V. Vapnik, *The nature of statistical learning theory*, Springer, New York, 1995.
- [Var02] A. Varki (ed.), *Essentials of glycobiology*, Cold Spring Harbor Laboratory Press, 2002.
- [Ver02] J.-P. Vert, *A tree kernel to analyze phylogenetic profiles*, Bioinformatics **18** (2002), S276–S284.
- [VS02] S. V. N. Viswanathan and A. J. Smola, *Fast kernels for string and tree matching*, Neural Information Processing Systems (NIPS2002), MIT Press, 2002, pp. 569–576.
- [WF74] R. A. Wagner and M. Fischer, *The string-to-string correction problem*, Journal of the Association for Computing Machinery (J. ACM) **21** (1974), no. 1, 168–173.
- [WZ01] J.T.-L. Wang and K. Zhang, *Finding similar consensus between trees: an algorithm and a distance hierarchy*, Pattern Recognition **34** (2001), 127–137.
- [WZ03] L. Wang and J. Zhao, *Parametric alignment of ordered trees*, Bioinformatics **19** (2003), no. 17, 2237–2245.
- [WZ05] L. Wang and K. Zhang, *Space efficient algorithms for ordered tree comparison.*, Proc. of 16th International Symposium of Algorithms and Computation (ISAAC), Lecture Notes in Computer Science, vol. 3827, 2005, pp. 380–391.
- [XRD⁺05] B. Xia, J. A. Royall, G. Damera, G. P. Sachdev, and R. D. Cummings, *Altered o-glycosylation and sulfation of airway mucins associated with cystic fibrosis*, Glycobiology **15** (2005), no. 8.
- [Yan91] W. Yang, *Identifying syntactic differences between two programs*, Software - Practice and Experience **21** (1991), no. 7, 739–755.
- [YKT05] R. Yang, P. Kalnis, and A. K. H. Tung, *Similarity evaluation on tree-structured data*, Proc. of the 2005 ACM SIGMOD international conference on Management of data, 2005, pp. 754–765.
- [ZAR03] D. Zelenko, C. Aone, and A. Richardella, *Kernel methods for relation extraction*, Journal of Machine Learning Research **3** (2003), 1083–1106.
- [Zha95] K. Zhang, *Algorithms for the constrained editing distance between ordered labeled trees and related problems*, Pattern Recognition **28** (1995), no. 3, 463–474.
- [Zha96] K. Zhang, *A constrained edit distance between unordered labeled trees*, Algorithmica **15** (1996), 205–222.
- [ZJ94] K. Zhang and T. Jiang, *Some MAX SNP-hard results concerning unordered labeled trees*, Information Processing Letters **49** (1994), no. 5, 249–254.
- [ZL05] Y. Zhai and B. Liu, *Web data extraction based on partial tree alignment*, Proc. of 14th International World Wide Web Conference (WWW2005), 2005, pp. 76–85.
- [ZS89] K. Zhang and D. Shasha, *Simple fast algorithms for the editing distance between trees and related problems*, SICOMP **18** (1989), no. 6, 1245–1262.
- [ZSS92] K. Zhang, R. Statman, and D. Shasha, *On the editing distance between unordered labeled trees*, Information Processing Letters **42** (1992), no. 3, 133–139.
- [ZSW94] K. Zhang, D. Shasha, and J. T.-L. Wang, *Approximate tree matching in the presence of variable length don't cares*, Journal of Algorithms **16** (1994), no. 1, 33–66.

Index

Symbols

$(\uparrow x)_T$	20
$(\downarrow x)_T$	20
\leq	19
\preceq	20
\trianglelefteq	24
$a \mapsto b$	11
$\mathcal{A}(S, T)$	39
$\mathcal{A}(x, y)$	14
\mathcal{C}^*	93
\mathcal{C} -mapping	32
$\text{ch}(y)$	20
$d_i(x, y)$	26
$\mathbf{D}^{\text{ALN}}(x, y)$	14
$\mathbf{D}^{\text{ALN}}(S, T)$	39
$\mathbf{D}_I^{\mathcal{C}}(S, T)$	38
$\mathbf{D}_{\text{ICS}}^{\mathcal{C}}(S, T)$	38
$\mathbf{D}_I^{\text{EDIT}}(x, y)$	11
$\mathbf{D}_q^{\text{GRAM}}(x, y)$	18
$\mathbf{D}_{\text{ICS}}^{\text{EDIT}}(x, y)$	12
$\text{deg}(x)$	20
$\text{deg}(T)$	20
$\delta(x, y)$	104
$\text{dep}(x)$	20
$\text{dep}(T)$	20
ε	10, 25
$E(T)$	20
\emptyset	22
$\mathcal{E}(S, T)$	26
$\mathcal{E}(x, y)$	11
\mathcal{F}	20
$F(v)$	21
\mathcal{F}_0	23
\mathcal{F}_N	121
$ F $	20
$l(x)$	21
$\text{LCS}(x, y)$	12
$ll(i)$	28
$\log^* n$	19
$M(x)$	32
$M^{-1}(x)$	32
$M[\cdot, \cdot]$	116
M_{ST}	32
$M^{(1)}$	32
$M^{(2)}$	32
$M_{23} \circ M_{12}$	33
$\mathcal{M}(S, T)$	32
$\mathcal{M}^{\mathcal{C}}(S, T)$	32

$\mathcal{M}(x, y)$	16
\mathbb{N}	4
$\text{par}(x)$	20
\mathbb{R}	4
\mathbb{R}_0^+	4
$rl(t)$	47
$\mathbf{R}_M(\cdot)$	47
$\mathbf{R}_M^{-1}(\cdot)$	50
Σ	10, 21
\mathcal{T}	20
$ T $	20
$T[U]$	21
$T \in F$	23
\mathcal{T}_0	23
\mathcal{T}_N	121
$UP_r(v)$	132
$V(T)$	20
$\ x\ _1$	4
$ x $	10
$x \mapsto y$	25
#P-complete	107
-	11, 25

A

absolute approximation	56
absorbent	121
Acc	91, 92
accordant mapping	92, 108
affine gap penalty	63
alignable mapping	39, 46, 85, 85 , 88
aligned tree	39, 85
alignment	13
alignment distance	14, 39
alignment of strings	13
alignment of trees	38
alignment problem	39
alphabet	10, 21
ancestor	19
approximate common subforest	36
approximate common subsequence	12
approximate common supersequence	13
approximate common supertree problem	41
area under the ROC curve	138

B

block move	19
bottom-up distance	61
bottom-up mapping	61

C

\mathcal{C} -distance 33
 chain 19
 child 20
 class label 101
 closure 93
 closure of tree mapping 93
 collapsed depth 29
 common subtree pattern 79, 80
 comparable 19
 complete subforest 21
 complete subtree 21
 composition of tree mappings 33
 concatenation 10
 constrained distance 2, 51
 constrained edit distance mapping 52
 constrained mapping 51, 81, 82, 83, 85
 constrained tree inclusion 62
 contraction 74
 convex gap cost 63
 convolution kernel 101, 102

D

decision function 101
 declarative definition 10, 16, 32
 declarative semantics 32
 decomposition strategy 26, 30
 degree 20, 45
 deletion 10, 78
 depth 20
 depth sequence 125
 descendent 19
 direction 30
 distance 9
 duality theorem 79
 duplication 74
 dynamic programming 14

E

edit graph 15, 59
 edit operation 10
 edit script 11, 26
 edit signature 10, 25
 elastic tree kernel 107
 embedding 70, 121
 Exact Cover by 3-Sets 31

F

feature 102
 feature score 140
 feature space 102
 fixed parameter algorithm 61
 fixed-parameter 16
 fixed-parameter algorithm 27, 41, 45
 forest 20
 induced by a set of nodes 21

G

gap 63

gap symbol 11, 13, 25
 gapped edit distance 63
 glycan 101, 137
 gram distribution 132
 gram distribution kernel 132
 Gram matrix 102

H

height 20
 hierarchical order 19, 20, 21, 68, 72

I

identity edit operation 12, 13, 36
 inclusion 62
 incomparable 19
 insertion 10, 73
 isolated-subtree distance 51
 isolated-subtree mapping 51
 isomorphism 69

K

kernel function 101, 102
 kernel method 101, 102
 kernel trick 102
 keyroots(T) 29
 Kronecker's delta 104

L

ℓ_1 -norm 4
 label sequence 24, 125
 labeled forest 21
 labeled tree 21
 labeled tree kernel 106
 labeled trees 21
 labelling function 21
 largest common subforest pattern 38
 largest common substructure 38
 largest common subtree 59
 layered trimer kernel 137
 LCA-preserving 76
 LCS cost 38, 59
 LCS problem 12, 16
 LCST 59
 leaf 20
 least common ancestor 72, 76
 leaves(T) 20
 left-total 62
 leftmost leaf 70
 less-constrained distance 2, 55
 less-constrained mapping 55, 84, 84, 85, 88
 Levenshtein distance 11
 LIBSVM 138
 line tree 126
 linear extension 24
 linear order 19
 local alignment 63
 local similarity 63
 longest common subsequence problem 12

low-distortion embeddings 18, 64
 Lu distance 50
 Lu mapping 92

M

match 126
 MAX SNP-hard 31, 90
 Maximum Bounded Covering by 3-Sets 31
 maximum weighted clique 31
 $maxpath(T)$ 132
 MDS 18
 merge operation 62
 metric 95
 metric space 9
 minimal 19
 minimum 19
 monotonic 33, 95
 motif 140
 move operation 19, 62
 multi-dimensional scaling 18
 multiple alignment 17
 multiple tree matching problem 61

N

n -gram 17
 node 19
 null node 25
 null string 10

O

occurrence 17, 126
 operational definition 9, 14, 26
 operational semantics 26
 optimal alignment 13, 14, 39
 optimal \mathcal{C} -mapping 33
 optimal edit script 11, 26
 ordered forest 20, 22
 ordered tree 20

P

parameterized complexity 16
 parent 20
 parent sequence 125
 parse tree kernel 106
 partial order 19
 partially ordered set 19
 poset 19
 positive semidefinite 102
 postorder 23, 125
 pq -gram 64
 preorder 23
 proper ancestor 19
 proper descendent 19
 proper subclass 32
 proper superclass 32
 pseudometric 9, 18

Q

q -gram 17, 126

q -gram distance 17, 63, 131
 q -gram profile 17, 105
 q -spectrum kernel 105, 131

R

relevant forest 28
 replacement 10
 residue 71
 right 20
 right branch 128
 root 19
 $root(T)$ 19
 root image 47, 50
 rooted ordered tree 20
 rooted tree 19

S

semi-accordant mapping 91
 sibling 20
 sibling order 20
 similarity function 102
 spectrum kernel 105, 125
 string 10
 string edit problem 11
 string kernel 104
 strongly structure-preserving mapping 50, 83
 strongly-preserving mapping 50
 structure-preserving distance 47
 structure-preserving mapping 2, 48, 82
 structure-respecting distance 55
 structure-respecting mapping 55, 81
 subadditive 9
 subclass 32
 subforest 21
 subsequence 10
 substring 10
 substring move 19
 subtree 21
 subtree pattern 21
 subtree-congruent mapping 106
 sugar chain 137
 superclass 32
 supersequence 13
 supertree 40
 support vector machine 101
 SVM 101
 swapping operation 62
 symmetric 33
 symmetry 95

T

Tai distance 2, 26
 Tai edit problem 26
 Tai mapping 34, 79, 114
 top-down common subtree isomorphism 59
 top-down distance 57
 top-down mapping 57, 106
 total order 19

totally ordered set	19
trace	16
traceback	14
training data	101
transitive	33, 35, 95, 121
transitivity	52
tree	20
tree q -gram	126
tree edit distance	2
tree edit problem	26
tree homomorphism	68, 70
tree inclusion	62
tree isomorphism	69
tree mapping	2, 32
U	
<hr/>	
unit costs	11, 38
unit-cost edit distance	11
universal tree	121
unordered forest	20
unordered tree	20
unordered trees	20
unrooted ordered tree	26
V	
<hr/>	
vertex	19
VLDC	31
W	
<hr/>	
weight vector	101
X	
<hr/>	
XML	62