Efficient Training of Observable Operator Models using Context Graphs

Tobias G. Oberstein

Master Thesis

Institute for Autonomous intelligent Systems, Fraunhofer AiS
Mathematical Institute / ZAIK, University Cologne
November 2002

Advisors

1. Advisor
Prof. Dr. Rainer Schrader
Zentrum für Angewandte Informatik Köln
Universität Köln

2. Advisor
Dr. Herbert Jaeger
Institut für Autonome intelligente Systeme
Fraunhofer AiS

Declaration

This thesis is based on work carried out at the Institute for Autonomous intelligent Systems, Fraunhofer AiS, Germany. No part of this thesis has been submitted elsewhere for any other degree or qualification and all work was done by myself unless referenced to the contrary in the text.

Tobias G. Oberstein (Matr. Nr. 3241122)

Abstract

Observable Operator Models (OOMs) are generative systems which can model stochastic time-series data and sequences. Training OOMs amounts to estimating linear operators from time-series sample data. The problem is that usually both the amount of available training data and our computational resources are restricted. The main contributions of this thesis are: (1) adaptation of tools from information theory to analyze OOM estimation from finite data, (2) development of a new data structure, *context graphs*, to effectively exploit structure inherent in the training sample for model estimation and (3) design and implementation of a software package for the simulation, analysis and training of OOMs.

Key words. observable operator model, hidden markov model, stochastic process, context graph, suffix tree, time series, nonlinear time series analysis, stochastic modeling, machine learning

AMS subject classifications. 37M10, 60G25, 62M09, 62M10, 62M20, 68P05

Zusammenfassung

Observable Operator Models (OOMs) sind generative Systeme die stochastische Zeitreihen und Sequenzen modellieren können. Das Training von OOMs besteht in der Schätzung von linearen Operatoren aus Zeitreihenstichproben. Das Problem hierbei ist, dass in der Regel sowohl die verfügbare Menge an Trainingsdaten als auch an Rechenresourcen beschränkt ist. Die wesentlichen Beiträge dieser Arbeit sind: (1) Anpassung von Werkzeugen aus der Informationstheorie zur Analyse der Schätzung von OOMs aus endlichen Daten, (2) Entwicklung einer neuen Datenstruktur, *Context Graphs*, zur effektiven Verwertung der inhärenten Struktur in Trainingsdaten zur Modellschätzung und (3) Design und Implementierung eines Softwarepaketes für die Simulation, Analyse und das Training von OOMs.

Schlüsselwörter. Observable Operator Model, Hidden Markov Model, stochastischer Prozess, Context Graph, Suffix Tree, Zeitreihe, nichtlineare Zeitreihenanalyse, stochastische Modellierung, Machine Learning

AMS Sachgebiete. 37M10, 60G25, 62M09, 62M10, 62M20, 68P05

Acknowledgements

First and foremost I am deeply grateful to Dr. Herbert Jaeger, Institute for Autonomous intelligent Systems, Fraunhofer AiS, for introducing me to OOMs, for his constant guidance, encouragement and many insightful discussions during the course of my study. Working in his team was demanding at times, always great fun and offered many chances to learn how research works and to acquire valuable know-how.

I am sincerely grateful to Prof. Dr. Rainer Schrader, Zentrum für Angewandte Informatik Köln (ZAIK), Universität Köln, for his advisory. I greatly appreciate his support, without this thesis would have been impossible.

Further, I am deeply indebted to Dr. Klaus Kretzschmar, Institute for Autonomous intelligent Systems, Fraunhofer AiS, for uncountable hours of fruitful and exciting discussions, always taking time explaining things to me and both his professional and personal support. This has been a great time.

Special thanks go to Dipl. Math. Alexander Schönhuth for his support and for looking over and discussing draft versions of this thesis.

Dedicated to my grandpa,
Alfred Sonnenberg.

Contents

Introduction

Observable Operator Models (OOMs) are generative systems which can model stochastic time-series data and sequences ([**Jae00**]). What follows is an informal introduction to OOMs and an outline and motivation of important topics treated in this thesis.

Time-series capture time dependent, time varying aspects of reality in series of measurements. At each instance of time, the aspect to be captured is sampled on some scale yielding a number. A *stochastic* time-series is supposed to be either generated by a process inherently non-deterministic, deterministic but scrambled with noise or at least to appear non-deterministic because of our lack or will of precise measurements[1]. Examples are weather data like temperature patterns, financial market data like stock quotes, speech and voice data and distorted or very complex sensory signals.

On the other hand, one often speaks of (stochastic) sequences when the aspect of reality captured has no obvious scale or when ordering (of measurements) is important but cannot be interpreted as "time" in the common sense. An example of the former is a natural language text, since the alphabet is finite and has no "scale". An example of the latter is a biosequence like DNA since the position of a symbol on a DNA strand has no direct interpretation as an instance in time.

One may look at stochastic time-series and sequences in terms of the probabilities of their occurrence. From this perspective a model is given by a collection of probabilities for all possible realizations one could observe. Chapter 1 will give precise definitions starting from the notion of stochastic processes.

Now, OOMs are models of stochastic time-series and sequences in the sense that they are formalized, compressed descriptions of the probability distributions of those series and sequences. OOMs are generators of stochastic time-series and sequences in the sense that they also constitute a stochastic mechanism for producing those series and sequences.

OOMs build a bridge between linear algebra and the theory of stochastic processes. They do this by describing the change of our knowledge about the future of a stochastic system by means of linear operators selected on the basis of present observations. Our knowledge about the future of a stochastic system is captured by a conditional distribution which specifies the probabilities of possible future developments given a particular, already realized past. This knowledge about the future of the system is subject to change when we observe a new present outcome. Hence, the newly arrived information must be reflected in our knowledge about the process future. Updating our knowledge about the process future is done by applying a linear operator selected from a finite fixed set of operators on the basis of the present outcome. A detailed introduction to OOMs from this prespective is given in chapter 2.

Any stochastic process can be described using OOMs when we allow for OOMs of infinite dimension. On the other hand, OOMs with *finite* dimension can still describe a broad class of stochastic processes. Indeed, the class of processes thus describable is a strict superset of those which can be captured by Hidden Markov Models with finitely many hidden states ([**Jae00**]). OOMs with finite dimension have some nice properties (see chapter 3). First, their defining linear operators can be written as matrices, which allows one to apply all the standard tools known from linear algebra. Second, we can decide when two OOMs define the same stochastic process, that is when they are equivalent and thereby see how equivalence classes of OOMs arise.

Model training in the context of OOMs means estimating linear operators from time-series sample data. How to estimate the matrices representing operators of finite dimensional OOMs from sample data is presented in chapter 5. The procedure makes use of so-called *indicative events* and *characteristic events* which can be thought of as rasters through which the sample is exploited. Those events also have a fundamental meaning to *interpretable* OOMs (see chaper 4) which are special OOMs within each equivalence class of OOMs. Importantly, the basic learning algorithm described is asymptotically correct.

[1]Formally, the line of distinction can be drawn quite accurately: inherent stochastic time-series are realizations of stochastic processes, whereas pseudo-stochastic time-series are discretized trajectories of dynamical systems in a chaotic regime

That is given infinite training data in the limit, the OOM estimated from the sample is almost always right. Moreover, this holds true independent of particular choices of indicative and characteristic events, which one is formally free to choose with the basic learning algorithm.

However, the situation with *finite* training data is fundamentally different. This is also the original motivation for the present thesis. With finite training data, the particular choice of indicative and characteristic events is most important as it will determine the quality of the estimated model. The basic learning algorithm is asymptotically correct, but nothing is said about the speed of convergence or possible bounds of model quality for finite training data, both of which *are* dependent on the particular choice of indicative and characteristic events used in estimating an OOM.

The thesis at hand provides first steps, methods and foundations towards a systematic treatment of *optimal and efficient estimation of OOMs from finite data*. "Optimal" since the amount of available training data is finite in practice and "efficient" since computational resources available in estimating the model are usually restricted. Three important subgoals have been identified. First, a measure of quality for estimated OOMs is needed, otherwise a method of "optimal estimation" can not be developed. Second, a data structure is needed which efficiently exploits the structure in the training sample, allows to represent candidate choices of indicative and characteristic events and facilitates the computation of counting statistics needed for the basic learning algorithm. Third, a theory is needed that allows to derive results for choosing optimal indicative and characteristic events.

The first subgoal, defining a rigorous measure of model quality is developed in detail throughout part 2 of this thesis. Fundamental notions from information theory have been customized and applied to measure quality of estimated OOMs and experiments were done to verify the dependency of model quality on training sample size and the particular choice of indicative and characteristic events.

The second subgoal has been met in part 3 by developing a new data structure, *context graphs* which build on suffix trees and are capable of representing and indexing all variable length contexts within a finite sequence. Further it is shown how to represent all candiate choices of indicative and characteristic events as special colorings of the context graph of the training sample. This is crucial to efficiency since it allows to represent the training sample and all candidates for indicative and characteristic events, that is rasters to exploit the sample within one unified data structure.

Context graphs also provide means to control the complexity of the rasters used in exploiting the training sample. This is of value, since the third subgoal likely involves finding a balance between overfitting and underexploiting the sample data. In general, under the constraint of finite sample data one must strive to fully exploit the sample achieving good model precision while not overfitting the sample maintaining good generalization performance. One may speculate, that an extensive treatment of this idea when applied to OOMs could require research along lines similar to those outlined in statistical learning theory ([**Vap98**], [**Vap99**]). Statistical learning theory explicitely and deeply addresses the problem of optimal model estimation from finite data, but only for static patterns and not time-series. Consequently, it might be interesting to apply and customize methods from statistical learning theory to optimal OOM estimation. However, due to time and space constraints this last approach could not be undertaken. Thus, regarding the third subgoal the thesis provides only first, but nevertheless valuable insights.

Finally, part 4 contains a detailed description and discussion of the software developed in the course of this thesis, the *Observable Operator Modeling Kit*. The software consists of $30k$ lines of C++ code and provides functionality for the simulation, analysis and training of OOMs. Also it contains and uses an implementation of the *context graph* data structure newly introduced in this thesis. The development of such a package was a major goal of this thesis besides providing theoretical results and a significant part of the work done in this thesis.

Part 1

Theory of Observable Operator Models

CHAPTER 1

Stochastic Processes

Stochastic modeling of sequence data and time-series relies on the notion of *stochastic processes*. This notion connects two fundamental concepts: time and random entities. Hence, due to their fundamental nature it should come as no surprise that stochastic processes reappear in the theory of information and communication under a different interpretation as *information sources*.

1.1. Random Variables

Random variables formalize the idea of entities dependent on random events. The term "random variable" is sometimes recepted as misleading, since its semantics is nowhere near those encountered in other fields for the term "variable". This trap for intuition may be avoided by thinking of random variables as what they are: measurable functions on probability spaces.

random variable

DEFINITION 1.1.
Let $(\Omega, \mathcal{F}, \mathsf{P})$ be a probability space and $(E, \mathcal{E})$ be a measurable space. A *random variable* X is a function

$$X : \Omega \longrightarrow E$$

which is measurable with respect to $\mathcal{F}$ and $\mathcal{E}$, e.g.

$$\forall M \in \mathcal{E} \ : \ \{\omega \in \Omega : X(w) \in M\} \in \mathcal{F}$$

Random variables canonically transport the probability measure P on the measurable space $(\Omega, \mathcal{F})$ to a probability measure P' on the measurable space $(E, \mathcal{E})$ by

$$\forall M \in \mathcal{E} \ : \ \mathsf{P}'(M) \ := \ \mathsf{P}(\{\omega \in \Omega : X(w) \in M\}) \tag{1.1}$$

Because of 1.1 sometimes $(E, \mathcal{E}, \mathsf{P}')$ is treated as a probability space itself without mentioning the primal probability space $(\Omega, \mathcal{F}, \mathsf{P})$.

1.2. Stochastic Processes

As previously stated, there are two components to stochastic processes: time and random entities. The latter is formalized by means of random variables. The former, the concept of time, as intricate it may seem philosophically and certainly is in everyday life, as simple it is formalized in mathematics. Time is an infinite and totally ordered set.

stochastic process

DEFINITION 1.2. A *stochastic process* is a family of random variables $(X_t)_{t \in T}$ all of which are defined on a common probability space $(\Omega, \mathcal{F}, \mathsf{P})$ and all are mapping into a common measurable space $(E, \mathcal{E})$ where T is a totally ordered set - the time set.

This notion leaves great freedom in choosing the individual random variables constituting the random process. Of course the more interesting cases are those where the random variables are somehow systematically related to one another.

Usually, a stochastic process is called *discrete time* if $T = \mathbb{N}$ or $T = \mathbb{Z}$. In the case of $T = \mathbb{R}^+$ or $T = \mathbb{R}$ the process is called *continuous time*.

realization
trajectory

Every function $T \ni t \mapsto X_t(\omega)$ for a fixed ω is called *realization* or *trajectory* of the stochastic process.

A stochastic process is called *finite valued* if E is finite, *discrete valued* if E is countable and *continuous valued* if E is not countable, e.g. $E = \mathbb{R}$.

In this thesis we will exclusively be concerned with discrete time stochastic processes that take values in a finite set called the alphabet, that is discrete time, finite valued stochastic processes.

1.3. Events in Stochastic Processes

For ease of reference to events happening in a stochastic process within certain time ranges let

$$\mathcal{F}_{[r,s]} = \sigma(\{X_t : r \leq t \leq s\}) \subset \mathcal{F}, \quad r, s \in T \tag{1.2}$$

denote the σ-(sub-)algebra that is generated by events of the form

$$\{X_t \in M_t : r \leq t \leq s,\ M_t \in \mathcal{E}\} \ \in \bigotimes_{r \leq t \leq s} \mathcal{E}_t \quad \text{where} \quad \mathcal{E}_t = \mathcal{E} \tag{1.3}$$

Here, $\bigotimes \mathcal{E}_t$ denotes the product σ-algebra over $\mathcal{E}$ and the range $[r, s]$. We also use $\mathcal{F}_{(-\infty,s]}$ and $\mathcal{F}_{[r,\infty)}$ in a similar interpretation. The former will allow us to refer to arbitrary events in the past of the stochastic process with respect to time s and the latter to refer to arbitrary events in the future of the stochastic process with respect to time r.

1.4. Characterizing Stochastic Processes

A stochastic process is defined by specifying the primary probability space $(\Omega, \mathcal{F}, \mathsf{P})$ plus the family of random variables $(X_t)_{t \in T}$ mapping into $(E, \mathcal{E})$.

Of course one may insist that choosing a specific set Ω is arbitrary up to the cardinality of Ω at least from an extensional perspective. Also, often we want to choose the σ-algebra of events $\mathcal{F}$ as big as possible, e.g. as the powerset of Ω or if this is not possible as some Borel σ-algebra over Ω.

Obviously, choosing P is much more important. In practice, one often simplifies things even further by forgetting about the primal probability space $(\Omega, \mathcal{F}, \mathsf{P})$ altogether and instead defines a probability measure P' on the product space

$$(\bigotimes_{t \in \mathsf{T}} E_t, \bigotimes_{t \in \mathsf{T}} \mathcal{E}_t) \quad \text{where} \quad E_t = E,\ \mathcal{E}_t = \mathcal{E} \tag{1.4}$$

As it turns out, defining a stochastic process as "a sequence of random variables on a perhaps very complicated underyling probability space" or "as a probability measure directly on the measurable space of possible output sequences" is equivalent at least in the case of discrete valued and discrete time stochastic processes [**Gra90**]. Let me note, that there is an interesting third formulation of stochastic processes in terms of dynamical systems and measurable transformations which is strictly more expressive and standard within the context of ergodic theory [**Gra90**]. And, as we will see, Observable Operator Models give us yet another representation of stochastic processes by means of linear algebra.

For the moment we note, that for discrete time and finite valued stochastic processes, which is the only species we deal within this thesis, the approach of defining the process by giving a measure on the output sequence space is especially convenient.

As E is finite, there is no problem in choosing the powerset σ-algebra $\mathcal{E} = \mathcal{P}(E)$. Without loss of generality, suppose $\mathsf{T} = \mathbb{N}$. Then, for defining a probability measure P' on $\bigotimes \mathcal{E}$ it suffices to define P' on every cylinder $C_{\bar{a}}$ where $\bar{a} \in E^n$

$$C_{\bar{a}} \;=\; \{\omega = (x_1, \ldots, x_n, x_{n+1}, \ldots) \in E^\infty : x_1 = a_1, \ldots, x_n = a_n\} \tag{1.5}$$

Literally, $\mathsf{P}'(C_{\bar{a}})$ is the probability for observing the inital sequence $\bar{a} \in E^n$. In the rest of the text, we will use the following cleaner notation

$$\mathsf{P}'(\bar{a}) \;=\; \mathsf{P}'(\{X_1 = a_1, \ldots, X_n = a_n\}) \;=\; \mathsf{P}'(C_{\bar{a}}) \tag{1.6}$$

In other words, the process is already defined by its distribution on finite inital sequences, that is the finite dimensional marginal distributions on inital sequences.

One remark on notation - starting from now we will write Σ instead of E for the finite set the finite valued processes take values in, Σ^n to denote all strings of length n and Σ^* or Σ^+ to denote the set of all finite strings including or excluding the empty string.

CHAPTER 2

Predictor Space OOMs

This section presents the category of OOMs as one specific way of defining general discrete time finite valued stochastic processes. Then, concrete OOMs are defined as one possible model of the OOM category.

This is probably neither the simplest nor the fastest route to an understanding of concrete OOMs but arguably the one which allows the deepest understanding.

The section closely follows the exposition presented in the original works [**Jae00**], [**Jae99**] of the OOM inventor and does not introduce anything new.

Note, that the restriction to discrete time finite valued stochastic processes is specific to this thesis. OOMs can likewise and similarily be defined for continuous time arbitrary valued stochastic processes [**Jae01**], [**Jae99**].

2.1. Representing Stochastic Processes by Conditional Distributions

The last section closed by stating that a discrete time, finite valued stochastic process $(X_t)_{t\in\mathbb{N}}$ can be fully specified by the probabilities of all finite initial sequences

$$\mathsf{P}(\bar{a}), \quad \bar{a} \in \Sigma^+ \tag{2.1}$$

We complete our notation by

$$\mathsf{P}(\bar{a} \mid \bar{b}) \;=\; \mathsf{P}(\{X_{l+1} = a_1, \ldots, X_{l+k+1} = a_k\} \mid \{X_1 = b_1, \ldots, X_l = b_l\}) \tag{2.2}$$

where $\bar{a} \in \Sigma^k$ and $\bar{b} \in \Sigma^l$.

Then of course (X_t) may likewise be characterized by all *conditional (continuation) probabilities*

conditional (continuation) probabilities

$$\mathsf{P}(\bar{a} \mid \bar{b}), \quad \bar{a} \in \Sigma^+,\ \bar{b} \in \Sigma^* \tag{2.3}$$

since the instances $\mathsf{P}(\bar{a}) = \mathsf{P}(\bar{a} \mid \epsilon)$ are trivially covered. While this may seem overly tedious compared to just giving probabilities for unconditioned initial sequences, it opens the door for introducing the key players in OOMs: linear operators.

Also note that $\mathsf{P}_{\bar{b}}(\cdot) = \mathsf{P}(\cdot \mid \bar{b})$ is a probability distribution on $\Omega = \Sigma^\infty$ for every fixed $\bar{b} \in \Sigma^*$. Similar to equation 1.5 we take

$$\mathsf{P}_{\bar{b}}(\bar{a}) = \mathsf{P}(w = (x_1, \ldots, x_l, x_{l+1}, \ldots, x_{l+k+1}, x_{l+k+2}, \ldots) \in \Sigma^\infty : \\ x_1 = b_1, \ldots, x_l = b_l, x_{l+1} = a_1, \ldots, x_{l+k+1} = a_k)$$

The distributions $\mathsf{P}(\cdot \mid \bar{b})$ are conditioned over $\bar{b}$. Hence, we may think of the conditional distribution $\mathsf{P}(\cdot \mid \bar{b})$ as describing the future of the process after an initial realization or specific past $\bar{b}$ of the process.

2.2. Conditional Distributions as Numerical Functions

For a deeper study of the conditional distributions that characterize a stochastic process introduced in the last section we will take things two steps further.

First, we will reformulate the conditional distributions as numerical functions which allows us to study those in an appropriate vector space of functions. Secondly, we will introduce linear operators on this vector space.

Intuitively, when retrofitted into the conditional distribution interpretation the linear operators will, given a specific past, model the change of knowledge about the future of the process when we observe a new present process output.

numerical predictor functions

Given $\mathsf{P}(\cdot|\ \bar{b})$, let

$$\begin{aligned} \mathfrak{g}_{\bar{b}} \ &: \ \Sigma^+ \to \ [0,1] \subset \mathbb{R} \\ \mathfrak{g}_{\bar{b}}(\bar{a}) &\mapsto \ \begin{cases} \mathsf{P}(\bar{a} \mid \bar{b}), \text{ if } \mathsf{P}(\bar{b}) > 0 \\ 0 \text{ else} \end{cases} \end{aligned} \tag{2.4}$$

Obviously, the set of numerical predictor functions $\{\mathfrak{g}_{\bar{b}} : \bar{b} \in \Sigma^*\}$ is yet another complete specification of the stochastic process.

We next study these functions within an appropriate real vector space. Let

$$\mathfrak{D} \ = \ \{\mathfrak{d} : \Sigma^+ \to \mathbb{R}\} \tag{2.5}$$

denote the set of real valued functions on finite, non-empty sequences. Then $\mathfrak{D}$ can be made into a real vector space by

$$(\alpha\mathfrak{d}_1 \ + \ \beta\mathfrak{d}_2)(\bar{a}) \ := \ \alpha(\mathfrak{d}_1(\bar{a})) \ + \ \beta(\mathfrak{d}_2(\bar{a})) \quad \forall \alpha, \beta \in \mathbb{R} \tag{2.6}$$

Within this vector space we can identify the linear subspace spanned by the numerical predictor functions representing conditional continuation distributions

$$\mathfrak{G} \ = \ \langle \{\mathfrak{g}_{\bar{b}} : \bar{b} \in \Sigma^*\} \rangle_{\mathfrak{D}} \tag{2.7}$$

2.3. Linear Operators on Conditional Distributions

The last piece missing from our initial outset are linear operators on the vector space $\mathfrak{G}$ we already have. For defining linear operators it suffices to specify the values they take on a basis of the involved vector space. Let $\Sigma_0^* \subset \Sigma^*$ such that $\{\mathfrak{g}_{\bar{e}} : \bar{e} \in \Sigma_0^*\}$ is a basis of $\mathfrak{G}$, not necessarily finite. Then define for every $a \in \Sigma$ a linear operator

$$\begin{aligned} \mathsf{t}_a \ &: \ \mathfrak{G} \to \mathfrak{G} \\ \mathsf{t}_a(\mathfrak{g}_{\bar{e}}) &:= \mathsf{P}(a \mid \bar{e}) \ \mathfrak{g}_{\bar{e}a} \end{aligned} \tag{2.8}$$

We have just built a bridge between the theory of stochastic processes and linear algebra. This will become clear from the propositions we now can prove.

PROPOSITION 2.1 (from [**Jae00**], [**Jae99**]). The constituting relation 2.8 for elements of the basis $\{\mathfrak{g}_{\bar{b}} : \bar{b} \in \Sigma_0^*\}$ carries over to the full vector space. For all $\bar{b} \in \Sigma^*$, $a \in \Sigma$ it holds that

$$\mathsf{t}_a(\mathfrak{g}_{\bar{b}}) = \mathsf{P}(a \mid \bar{b}) \ \mathfrak{g}_{\bar{b}a}$$

PROOF. Let $\bar{b} \in \Sigma^*$ and $\mathfrak{g}_{\bar{b}} = \sum_{i=1}^n \alpha_i \mathfrak{g}_{\bar{e}_i}$ be the linear combination of $\mathfrak{g}_{\bar{b}}$ from basis elements. Note that though the basis may be infinite, $\bar{b}$ always can be written as a linear combination of a finite number of basis elements. This is a characteristic of every vector space basis and a vector space has always a basis, at least if one accepts the axiom of choice. Let $\bar{c} \in \Sigma^+$. Then

$$\begin{aligned}
\mathtt{t}_a(\mathfrak{g}_{\bar{b}})(\bar{c}) &= (\mathtt{t}_a(\sum_{i=1}^{n}\alpha_i\mathfrak{g}_{\bar{e}_i}))(\bar{c}) \overset{\mathtt{t}_a \text{ is linear}}{=} (\sum_{i=1}^{n}\alpha_i\mathtt{t}_a(\mathfrak{g}_{\bar{e}_i}))(\bar{c}) \\
&\overset{\text{Def. of } \mathtt{t}_a}{=} (\sum_{i=1}^{n}\alpha_i\mathsf{P}(a|\bar{e}_i)\mathfrak{g}_{\bar{e}_i a}))(\bar{c}) \overset{\text{vector space linearity}}{=} \sum_{i=1}^{n}\alpha_i\mathsf{P}(a|\bar{e}_i)\mathsf{P}(\bar{c}|\bar{e}_i a) \\
&= \sum_{i=1}^{n}\alpha_i\mathsf{P}(a|\bar{e}_i)\frac{\mathsf{P}(\bar{e}_i a\bar{c})}{\mathsf{P}(a|\bar{e}_i)\mathsf{P}(\bar{e}_i)} = \sum_{i=1}^{n}\alpha_i\frac{\mathsf{P}(\bar{e}_i)\mathsf{P}(a\bar{c}|\bar{e}_i)}{\mathsf{P}(\bar{e}_i)} \\
&\overset{\text{lin. comb. } \mathfrak{g}_{\bar{b}}}{=} \mathfrak{g}_{\bar{b}}(a\bar{c}) = \mathsf{P}(a\bar{c}|\bar{b}) = \mathsf{P}(a|\bar{b})\mathsf{P}(\bar{c}|\bar{b}a) \\
&= \mathsf{P}(a|\bar{b})\mathfrak{g}_{\bar{b}a}(\bar{c})
\end{aligned}$$

□

What does this all mean? Well, the available knowledge about the future distribution of the process given the realization $\bar{b}$ is fully captured in the numerical predictor function $\mathfrak{g}_{\bar{b}}$ as it simply encodes the conditional continuation distribution $\mathsf{P}(\cdot|\bar{b})$. On the other hand $\mathfrak{g}_{\bar{b}a}$ captures the future distribution $\mathsf{P}(\cdot|\bar{b}a)$ of the process given realization $\bar{b}a$. The relation between these two predictors literally is the change of knowledge due to an observation of a after $\bar{b}$ was observed. Up to a scaling factor, this is precisely what the linear operator $\mathtt{t}_a$ does, not only on the elements of the basis but on all elements of the vector space, as the last proposition showed.

2.4. Computing Probabilities from Linear Operators

To fully prove the claim that the linear operators defined via 2.8 on the mentioned vector space completely describe the stochastic process we now show how to compute the correct probabilities of arbitrary finite initial sequences.

PROPOSITION 2.2 (from [**Jae00**], [**Jae99**]). Let $\{\mathfrak{g}_{\bar{e}} : \bar{e} \in \Sigma_0^*\}$ be a basis of $\mathfrak{G}$ and $\bar{a} = a_1 \dots a_k \in \Sigma^k$ an initial realization of the process. Then looking at $\mathtt{t}_{a_k} \dots \mathtt{t}_{a_1}\mathfrak{g}_\epsilon$ it holds that

$$\mathtt{t}_{a_k} \dots \mathtt{t}_{a_1}\mathfrak{g}_\epsilon = \sum_{i=1}^{n}\alpha_i\mathfrak{g}_{\bar{e}_i} \quad \Rightarrow \quad \mathsf{P}(\bar{a}) = \sum_{i=1}^{n}\alpha_i \tag{2.9}$$

In other words, the probability of an initial realization $\bar{a}$ can be computed as the sum of linear coefficients in the linear combination of $\mathtt{t}_{a_k} \dots \mathtt{t}_{a_1}\mathfrak{g}_\epsilon$ from basis elements.

PROOF. Iteratively applying 2.8 shows that $\mathtt{t}_{a_k} \dots \mathtt{t}_{a_1}\mathfrak{g}_\epsilon = \mathsf{P}(a_1 \dots a_k)\mathfrak{g}_{a_1 \dots a_k}$. Hence

$$\mathfrak{g}_{a_1 \dots a_k} = \sum_{i=1}^{n}\frac{\alpha_i}{\mathsf{P}(a_1 \dots a_k)}\mathfrak{g}_{\bar{e}_i}$$

Further, because the numerical predictor functions $\mathfrak{g}_{a_1 \dots a_k}$ and $\mathfrak{g}_{\bar{e}_i}$ describe probability distributions, the linear coefficients in the equation above must sum up to 1

$$\sum_{i=1}^{n}\frac{\alpha_i}{\mathsf{P}(a_1 \dots a_k)} = 1,$$

which immediately yields

$$\sum_{i=1}^{n}\alpha_i = \mathsf{P}(a_1 \dots a_k),$$

which had to be shown. □

2.5. From Linear Operators to Conditional Distributions

The last section constructed a "linear operators on vector space"- representation of stochastic processes by starting from a given process probability distribution. The converse will be described in this section. That is, necessary and sufficient conditions will be given such that a set of linear operators on a real vector space defines a stochastic process.

Given a real vector space V with a basis $(e_j)_{j\in J}$. Let $\sigma_{(e_j)_{j\in J}} : V \to \mathbb{R}$ be the numerical function such that

$$V \ni w = \sum_{i=1}^{k} \alpha_i e_{j_i} \quad \Rightarrow \quad \sigma_{(e_j)_{j\in J}}(w) = \sum_{i=1}^{k} \alpha_i \tag{2.10}$$

Thus, σ assigns to every vector $w \in V$ the sum of coefficients from the linear combination of w from basis vectors. Also note that σ itself is linear. In the following, we use the shorthand $\tau_{\bar{a}} = \tau_{a_k} \cdots \tau_{a_1}$ where $\bar{a} \in \Sigma^k$.

Proposition 2.3 (from [**Jae00**], [**Jae99**]). Let V be a real vector space with a basis $(e_j)_{j\in J}$, Σ a finite set, $(\tau_a)_{a\in\Sigma}$ a family of linear operators on V and $v_0 \in V$. Define

$$P : \Sigma^* \to \mathbb{R} \quad \text{by} \quad \mathsf{P}(\bar{a}) := \sigma(\tau_{\bar{a}} v_0), \quad \mathsf{P}(\epsilon) := 1$$

and $\mu := \sum_{a\in\Sigma} \tau_a$.

Then P can be extended to the distribution of a discrete time finite valued stochastic process iff the following three conditions are met

(1) $\sigma(v_0) = 1$

(2) $\sigma(\mu e_j) = \sigma(e_j)$ for all basis vectors e_j

(3) $\sigma(\tau_{\bar{a}} v_0) \geq 0$ for all $\bar{a} \in \Sigma^*$

Proof. $\Leftarrow$: A numerical function $P : \Sigma^* \to \mathbb{R}$ can be uniquely extended to the distribution of a discrete time finite valued stochastic process iff for all $n \geq 1$ the following three conditions are met

$$(a) \quad \mathsf{P}(a_1 \dots a_n) \geq 0$$

$$(b) \quad \sum_{a_1 \dots a_n \in \Sigma^n} \mathsf{P}(a_1 \dots a_n) = 1$$

$$(c) \quad \mathsf{P}(a_1 \dots a_n) \;=\; \sum_{b\in\Sigma} \mathsf{P}(a_1 \dots a_n b)$$

(a) follows from condition (3) of the above proposition. Condition (1) and (2) in the above proposition imply that $\sigma(\mu w) = \sigma(w)$ for all $w \in V$ and that

$$\Sigma_{a_1 \dots a_n \in \Sigma^n} \sigma(\tau_{a_n} \cdots \tau_{a_1} v_0) = \sigma(\mu \cdots \mu v_0)$$

from which we can see (b) and (c)

$\Rightarrow$: If P can be extended to a distribution, then conditions (1) and (3) are obviously fullfilled. Further, observe that $\sigma(\mu\tau_{\bar{a}} v_0) = \sigma(\sum_{b\in\Sigma} \tau_b \tau_{\bar{a}} v_0) = \sigma(\tau_{\bar{a}} v_0)$. Since V is spanned by the vectors $\{\tau_{\bar{a}} v_0 | \bar{a} \in \Sigma^*\}$, this implies that for all $w \in V$ it holds that $\sigma(\mu w) = \sigma(w)$, which subsumes condition (2) as a special case. □

2.6. The Dual Representations of Processes

The dual representation of stochastic processes as conditional (continuation) distributions and as linear operators on a real vector space of numerical predictor functions is at the core of observable operator theory. We summarize this in the following

Definition 2.1 (Predictor-Space OOM). Let $(X_t)_{t\in\mathbb{N}}$ be a stochastic process with values in a finite set Σ. The structure $(\mathfrak{G}, (\mathsf{t}_a)_{a\in\Sigma}, \mathfrak{g}_\epsilon)$ is called the *predictor space observable operator model* of the process. The vector space dimension of $\mathfrak{G}$ is called the dimension of the process and is denoted $\mathsf{dim}((X_t))$.

It is quite remarkable that besides the restriction to discrete time finite valued processes, which is merely a restriction of the scope of this thesis, we imposed no restriction on generality. We have arrived at a completely general characterization of discrete time, finite valued stochastic processes in terms of linear operators on real vector spaces of numerical predictor functions.

The reader may wonder how it can be that stochastic processes can always be characterized by *linear* operators. Of course this is the case only because we allowed vector spaces of infinite dimension. In the very moment we restrict ourselves to finite dimensional vector spaces, we cannot generally expect to be able to represent every stochastic process using linear operators on this vector space.

The good news is that a broad class of practically relevant situations may well be served by finite dimensional linear stochastic processes. Also it is important to recognize that the class of observable operator models is a strict superset of the popular hidden markov models. There are finite dimensional OOMs that can only be implemented by hidden markov models with infinitely many states. An example is the "probability clock" which is discussed in [**Jae00**].

CHAPTER 3

Concrete OOMs

The last section introduced observable operator models on a very abstract level. As it turns out, vector spaces of numerical predictor functions are not very handy in practice. Fortunately, we can use appropriate vector space isomorphisms to transfer our doing to the mundane world of some $\mathbb{R}^m$.

As with the last section, also in this section I reproduce the original work [**Jae00**], [**Jae99**] for the sake of a coherent and closed presentation.

3.1. Definition of Concrete OOMs

concrete OOMs

We now give the definition of *concrete* OOMs, which are algebraic structures directly corresponding and equivalent to the abstract OOMs introduced in the last section. The big advantage is that concrete OOMs are defined over the vector spaces $\mathbb{R}^m$, which immediately opens up the toolbox of linear algebra for working with concrete OOMs.

In the following we use $\mathbf{1} = (1, \ldots, 1) \in \mathbb{R}^m$ to denote the row vector consisting of all 1's.

DEFINITION 3.1. A m-dimensional OOM is a triple $\mathcal{A} = (\mathbb{R}^m, (\tau_a)_{a\in\Sigma}, w_0)$, where $w_0 \in \mathbb{R}^m$ and $\tau_a : \mathbb{R}^m \to \mathbb{R}^m$ are linear operators, satisfying

(1) $\mathbf{1}w_0 = 1$,
(2) $\mathbf{1}\mu = \mathbf{1}$, where $\mu = \sum_{a\in\Sigma} \tau_a$,
(3) $\forall a_1, \ldots, a_k \in \Sigma^k : \ \mathbf{1}\tau_{a_k} \cdots \tau_{a_1} w_0 \geq 0$.

Note, that the vector space dimension m of $\mathbb{R}^m$ directly corresponds to the dimensions in the abstract vector spaces of numerical predictor functions introduced in the previous section if the OOMs have minimal dimension and the three constituting conditions directly correspond to the three conditions given in proposition 2.3. As with abstract OOMs, the family of operators defining the OOM is indexed over the output alphabet of the stochastic process defined. By the way, this last correspondence is also the background that gave rise to the name "observable operator" models.

At this point it is worth noting, that the above definition is non-constructive because of (3). This has a couple of major implications and further analysis gives rise to a important strand in OOM research working with tools from the theory of convex cones. For the practical setting of estimating concrete OOMs from sample data, the problem is much less pressing and generally is manageable.

Now, without proof

PROPOSITION 3.1 (from [**Jae00**], [**Jae99**]). Let $\mathcal{A} = (\mathbb{R}^m, (\tau_a)_{a\in\Sigma}, w_0)$ be a concrete OOM, $\Omega = \Sigma^\infty$ and $\mathfrak{A}$ be the σ-algebra generated by all finite-length inital events on Ω. Then a numerical function

$$\mathsf{P}'(\bar{a}) \ := \ \mathbf{1}\tau_{\bar{a}} w_0$$

can be uniquely extended to a probability measure P on $(\Omega, \mathfrak{A})$ defining a discrete time, finite valued stochastic process $(\Omega, \mathfrak{A}, \mathsf{P}, (X_t))$.

Again, this directly mirrors proposition 2.3.

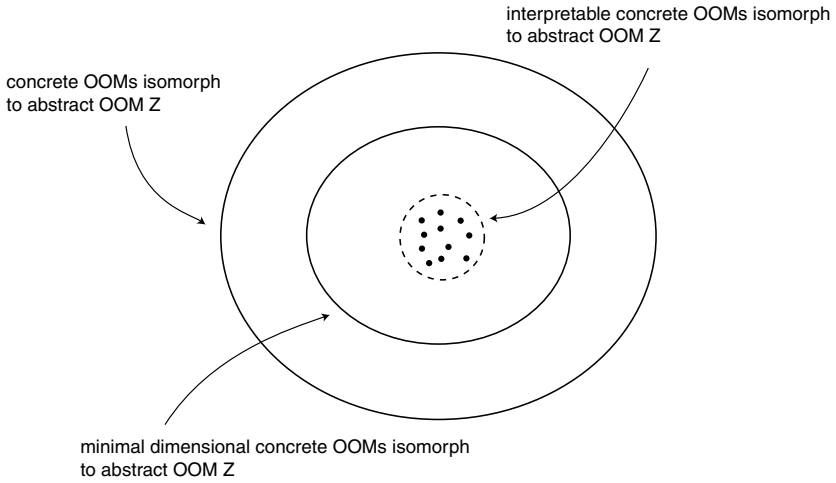

FIGURE 3.1. Class hierarchy of isomorphic concrete OOMs.

3.2. Equivalence of Concrete OOMs

vector space isomorphisms

The vector space isomorphisms between the vector spaces of numerical predictor functions and the vector spaces $\mathbb{R}^m$ are established via the following

PROPOSITION 3.2 (from [**Jae00**], [**Jae99**]). (1) If (X_t) is a discrete time, finite valued stochastic process describable by an m-dimensional abstract OOM, then there exists a m-dimensional concrete OOM describing the same process.

(2) If $\mathcal{A} = (\mathbb{R}^k, (\tau_a)_{a\in\Sigma}, w_0)$ is a k-dimensional concrete OOM, then there is exactly one abstract OOM of dimension $m \leq k$ describing the same process.

The proposition is significant, since it allows us to work with concrete OOMs instead of abstract OOMs.

An important point is that though abstract OOMs always have a one-to-one relation to the stochastic processes they define, for concrete OOMs this is not true as may be seen from (2) in above proposition. A concrete OOM may be given in a vector space of greater dimension that would be necessary, in which case the corresponding abstract OOM has a smaller dimension. But even if a concrete OOM has *minimal dimension* among those corresponding to the same abstract OOM, there still generally are uncountable many equivalent concrete OOMs in the equivalence class corresponding to a given abstract OOM. A sketch of the relations is given in figure 3.1. Given an abstract OOM there correspond classes of isomorphic concrete OOMs and minimal dimensional concrete OOMs, both of which generally are of uncountable infinite cardinality. Further a third class, *interpretable* OOMs, which we introduce in section 4 are distinguished. This class is of *countable* cardinality.

Given an OOM $\mathcal{A}$ it is possible to construct an equivalent minimal dimensional OOM $\mathcal{A}'$ [[**Jae00**]]. Based on this, the following proposition clarifies the necessary and sufficient conditions for two concrete OOMs to be equivalent in the sense of being isomorphic to the same abstract OOM.

PROPOSITION 3.3. Given two OOMs $\mathcal{A} = (\mathbb{R}^k, (\tau_a)_{a\in\Sigma}, w_0)$ and $\mathcal{B} = (\mathbb{R}^l, (\tau'_a)_{a\in\Sigma}, w'_0)$. Then $\mathcal{A}$ and $\mathcal{B}$ are equivalent (describe the same stochastic process) iff

(1) $\mathcal{A}$ and $\mathcal{B}$ have minimal dimensional OOMs of the same dimension m
(2) $\exists\rho : \mathbb{R}^m \to \mathbb{R}^m$ where ρ is linear such that
   (a) $\rho(w_0) = w'_0$,
   (b) $\tau'_a = \rho\tau_a\rho^{-1}$ for all $a \in \Sigma$,
   (c) $\mathbf{1}v = \mathbf{1}\rho v$ for all (column) vectors $v \in \mathbb{R}^m$

Starting from now, when not otherwise noted, we will exclusively work with concrete OOMs and omit the word "concrete".

Also, whenever we want to refer to a complete equivalence class of OOMs, we will use the following notion

linear dependent process

DEFINITION 3.2. A *linear dependent process* (LDP) of dimension $m \in \mathbb{N}$ is a stochastic process which can be described by a minimal m-dimensional concrete OOM.

Again, a LDP usually has (uncountable) many OOM representations and we can think of a LDP as a label of the particular equivalence class.

CHAPTER 4

Interpretable OOMs

In subsection 3.2 we saw that a given OOM is contained in a whole equivalence class of OOMs that induce the same stochastic process. Within such an equivalence class there is a proper subset of minimal dimensional OOMs which have a state space dimension equal to the dimension of the stochastic process defined by any OOM in the equivalence class (see figure 3.1). Also, we already distinguished a countable, proper subset within the minimal dimensional OOMs, the *interpretable* OOMs, but omitted a detailed exposition. This will be made up in the present section in which I paraphrase the results in [**Jae00**].

Interpretable OOMs are minimal dimensional OOMs with very special properties:

(1) their state vectors are probability vectors
(2) the components of their state vectors give probabilities of certain well-defined future events
(3) they can be constructively obtained through a "learning algorithm"

This section will discuss the properties (1) and (2), whereas property (3) will be discussed in the context of the learning algorithm presented in the next section.

4.1. State Vectors

Let $\mathcal{A} = (\mathbb{R}^m, (\tau_a)_{a\in\Sigma}, w_0)$ be a finite dimensional concrete OOM. Then the probability of observing an initial realization $\bar{b} = (b_1, \ldots, b_t) \in \Sigma^t$ starting from the state vector w_0 can be computed by

$$0 \leq \mathsf{P}(\bar{b}) = \mathbf{1}\tau_{\bar{b}} w_0 \leq 1 \tag{4.1}$$

Now, what about the probablity of observing $a \in \Sigma$ after having already observed $\bar{b}$? Of course, from above we could readily compute $\mathbf{1}\tau_{a\bar{b}}w_0$. However, suppose we know $\mathsf{P}(\bar{b})$, then we can also compute $\mathsf{P}(a\bar{b})$ *incrementally* since

$$\mathsf{P}(a|\bar{b}) = \frac{\mathsf{P}(a\bar{b})}{\mathsf{P}(\bar{b})} \;=\; \frac{\mathbf{1}\tau_{a\bar{b}}w_0}{\mathbf{1}\tau_{\bar{b}}w_0} \;=\; \mathbf{1}\tau_a \frac{\tau_{\bar{b}}w_0}{\mathbf{1}\tau_{\bar{b}}w_0} \;=:\; \mathbf{1}\tau_a w_t \tag{4.2}$$

and

$$w_t \;=\; \frac{\tau_{b_t} w_{t-1}}{\mathbf{1}\tau_{b_t} w_{t-1}} \tag{4.3}$$

The vectors $w_t \in \mathbb{R}^m$ the OOM passes through in the course of a certain trajectory are called *state vectors*. Note, that in general w_t will neither have column sum 1 nor non-negative components. The state vectors allow us to incrementally compute probabilities of observing sequences of but do not have a valid interpretation beyond this "helper role". However, for interpretable OOMs the state vectors will attain an additional important interpretation, namely as probability vectors which components give the probabilities of certain future events.

state vectors

4.2. Indicative and Characteristic Events

Events in stochastic processes were introduced in subsection 1.3, in particular equation 1.2 presented a notation for the σ-algebra subsuming events happening in a limited time window

$$\mathcal{F}_{[t,t+k]} \tag{4.4}$$

For discrete time ($T = \mathbb{N}$) stochastic processes (X_t) taking values in a finite alphabet Σ we may specify such events A_i by

$$(X_t, \ldots, X_{t+k}) \in A_i \subset \Sigma^k \tag{4.5}$$

Consequently, $\mathsf{P}((X_t, \ldots, X_{t+k}) \in A_i)$ denotes the probability of observing the process trajectory passing through A_i in the time window $[t, t+k]$. For convenience, we will use the shorthand $\mathsf{P}(A_i) := \mathsf{P}((X_1, \ldots, X_k) \in A_i)$.

Indicative and characteristic events now introduced are of the form we just encountered: events in a finite time window. Again, they are important for both theoretical and practical reasons

(1) for interpretable OOMs, the probabilities of characteristic events are given by the components of the OOM's state vectors
(2) indicative and characteristic events, once defined, can be used to estimate an interpretable OOM from sample data

Formally, indicative and characteristic events are defined

indicative events
characteristic events

DEFINITION 4.1. Let $\mathcal{L}$ be a linear dependent process of dimension m on the finite alphabet Σ with probability distribution P. Let $l, k \in \mathbb{N}$, $B_1 \dot\cup \ldots \dot\cup B_m = \Sigma^l$ with $B_j \neq \emptyset$ and $A_1 \dot\cup \ldots \dot\cup A_m = \Sigma^k$ with $A_i \neq \emptyset$ two partitionings of Σ^l and Σ^k into mutual disjoint, non-empty sets such that the matrix

$$V = (v_{ij})_{i,j \in \{1,\ldots,m\}} \quad \text{where} \quad v_{ij} \;=\; \mathsf{P}(A_i | B_j)$$

is nonsingular. Then the sets A_i are called *characteristic events* and the sets B_j are called *indicative events* of the process $\mathcal{L}$.

Note that here $\mathsf{P}(A_i|B_j) = \sum_{\bar a \in A_i,\ \bar b \in B_j} \mathsf{P}(\bar a | \bar b)$. The requirement of V being nonsingular looks more restrictive than it really is. In fact, in general it is more difficult to come up with indicative and characteristic events (where l and k are sufficiently large) that result in V being singular. In fact

PROPOSITION 4.1. Every finite dimensional LDP has characteristic and indicative events.

A proof may be found in [**Jae00**] (proposition 7).

Let me note that, despite the fact that the matrix V is either singular or not depending on what partitionings were chosen, from a numerical stand we may ask about the *extent of nonsingularity*, that is the numerical rank or condition number of V. In subsection 5.5 we will see, that for the practical problem of OOM estimation from finite data, the numerical rank of V is indeed of high importance. In other words, we are not only interested in partitionings such that V is formally nonsingular and thus the partitionings can be considered indicative and characteristic events, but rather in partitionings such that V has a condition number near 1 or a numerical rank near m.

4.3. Interpretable OOMs

We are now in a position to clearify our introductory comment, that interpretable OOMs will have the special property of giving the probabilities of certain future events directly as the component values of their state vectors. We will do this by first giving an abstract definition of interpretable OOMs and then show how to transform any OOM into an equivalent and interpretable OOM.

The "future events" we talked about are the characteristic events A_i and their probability of occurrence is given by the component values $(w_t)_i$ of the state vectors w_t. Formally,

interpretable OOM

DEFINITION 4.2. Let $\mathcal{A} = (\mathbb{R}^m, (\tau_a)_{a\in\Sigma}, w_0)$ be a finite dimensional concrete OOM and let $(A_i)_i$ and $(B_j)_j$ be characteristic and indicative events of $\mathcal{A}$. Then $\mathcal{A}$ is called *interpretable* with respect to the characteristic events $(A_i)_i$ if

$$\mathsf{P}(A_i \mid w_t) \;=\; (w_t)_i \quad \forall t \in \mathbb{N},\ \forall i \in \{1,\dots,m\}$$

Here, $\mathsf{P}(A_i \mid w_t)$ denotes the probability of observing $(X_{t+1},\dots,X_{t+k}) \in A_i$ given that the OOM was in state w_t at time t. Further, $(w_t)_i$ denotes the i-th component of the state vector w_t. Note, that since $A_1 \dot\cup \dots \dot\cup A_m$ is an exhaustive and disjoint partitioning of Σ^k it follows that $\sum_i \mathsf{P}(A_i \mid w_t) = 1$ and hence $\mathbf{1}w_t = 1$. In other words, w_t is a probability vector.

At this point, it is unclear why such OOMs should exist and also why every OOM should be equivalent to countable many interpretable OOMs. But this exactly is shown by the following

LEMMA 4.1. Let $\mathcal{A} = (\mathbb{R}^m, (\tau_a)_{a\in\Sigma}, w_0)$ be a minimal dimensional OOM and let $(A_i)_i$ and $(B_j)_j$ be characteristic and indicative events of $\mathcal{A}$. Define a linear mapping $\sigma : \mathbb{R}^m \to \mathbb{R}^m$ by

$$\sigma(w) := (\mathbf{1}\tau_{A_1} w, \dots, \mathbf{1}\tau_{A_m} w)$$

Then $\mathcal{A}' = (\mathbb{R}^m, (\sigma\tau_a\sigma^{-1})_{a\in\Sigma}, \sigma w_0)$ is an interpretable OOM equivalent to $\mathcal{A}$.

A proof may be found in [**Jae00**] (discussion in section 7). In the introduction to this section it was said that interpretable OOMs would be important both for theoretical and practical reasons. What we have just seen is probably mostly of theoretical interest. Yet from a practical perspective, the following proposition will lay the foundation for the learning algorithm introduced in the next section.

PROPOSITION 4.2. In an interpretable OOM (interpretable with respect to characteristic and indicative events A_i and B_j) it holds that

$$(1) \quad w_0 \;=\; (\mathsf{P}(A_1),\dots,\mathsf{P}(A_m))$$
$$(2) \quad \tau_{B_j} w_0 \;=\; (\mathsf{P}(B_j A_1),\dots,\mathsf{P}(B_j A_m))$$

Here, $\tau_{B_j} = \sum_{b\in B_j} \tau_b$. A proof may be found in [**Jae00**] (proposition 8).

CHAPTER 5

Learning OOMs

In this section we will see how to estimate an OOM from a given sample of a stochastic process or from stochastic time-series data. Basically, I follow and reproduce the results in the original works ([**Jae98**], [**Jae00**]) and only add to the discussion.

First, I outline the basic OOM learning algorithm, a purely mechanical and constructive procedure to compute an OOM when given a finite sample and when one has chosen a suitable target OOM dimension and suitable indicative and characteristic events.

The second subsection will present a result showing that the basic learning algorithm is an asymptotically correct method. The result holds true (largely) independent of the particular choice of indicative and characteristic events, but only in the limit case of infinite sample data.

The situation of *finite* sample data is fundamentally different and in fact the original motivation for this thesis. The rest of this thesis will then be devoted to applying and customizing different tools and developing first insights into the non-asymptotic learning situation.

5.1. The Sample

A sample is a finite string $s \in \Sigma^n$ that we are given and that is supposed to be produced by a process hidden from us.

Learning in the context of the present thesis means estimating or computing a model, specifically an concrecte OOM from the sample. However, for reasons of sanity we must make *some* assumptions about the unknown process behind the curtain that produced the sample. In particular we assume the sample to be generated by a

(1) linear dependent process with
(2) finite dimension, which is
(3) stationary and
(4) ergodic

(1),(2): A concrete OOM cannot represent anything "more" than a finite dimensional LDP. Hence it makes no sense to learn an OOM from a sample produced by a process that cannot be represented by an OOM. (3) This is a restriction of scope made in this thesis. (4) Is necessary since otherwise one cannot approximate probabilities via frequencies from a sample of a single realization in the limit (see section 8).

I should note that restriction (4) is not mentioned in the original work but introduced in this thesis and I will therefore shortly discuss the arguments that motivated me. I presume a situation in which it is either impossible to acquire or to make use of samples of more than a single process realization. For example, assume the sample is generated by a stationary real-world process. Then the finite sample we take will be a finite section of a *single* realization of the process. This is the case because the process both runs in real-time and is sampled in real-time. We cannot acquire sections from a second realization of the process since do not have access to the appropriate parallel universe. In other words, we are principally restricted to finite sections of a single realization of the process. Now on the other hand, assume we would be capable of acquiring a sample containing information of more than one realization of the stationary process. If the process is non-ergodic, in the

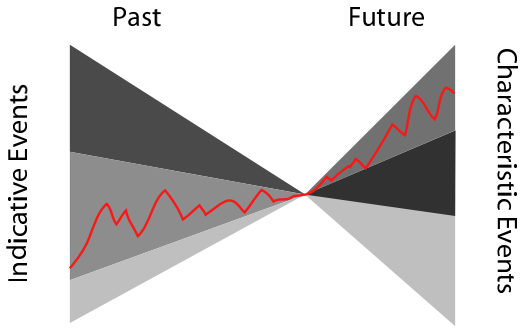

FIGURE 5.1. Sorting out a process realization into indicative and characteristic events.

worst case we needed a sample containing data from infinite many process realizations. However such a sample is no longer finite.

Beginning from now, whenever we speak of a sample, we refer to a sample assumed to be produced by a process a described above.

5.2. The Rasters of Indicative and Characteristic Events

When learning statistical models from data, one usually employs a certain stand: first, not every detail in the training data is considered meaningful due to the inherent stochastic nature of the process that generated the training data and second, generalization is desirable. These are good reasons for looking at a concrete realization of a stochastic process through some kind of raster, coarsening out the irrelevant.

Indeed, we already encountered such rasters in the form of indicative and characteristic events, introduced in definition 4.1 together with interpretable OOMs.

Indicative and characteristic events were defined by partitionings of the sequence space. Then, given a particular process realization, finite pieces of the realized trajectory into the past (*concrete past context*) and finite pieces of the realized trajectory into the future (*concrete future context*) relative to some fixed position in the realization (present) can be sorted out into disjunctive classes, the indicative and characteristic events (see figure 5.1).

concrete past context

concrete future context

The procedure of sorting out a process realization (sample) due to given rasters and counting the occurrences of the rasters is the basis for the basic OOM learning algorithm presented in the next section (in particular, see equations 5.5).

5.3. The Basic Learning Algorithm

The basic OOM learning algorithm is a purely mechnical and constructive procedure to compute an target OOM from the sample s, given the following additional input

(1) a dimension $m \in \mathbb{N}$ for the target OOM
(2) numbers $l, k \in \mathbb{N}$
(3) a partitioning $B_1 \dot\cup \ldots \dot\cup B_m = \Sigma^l$ with $B_j \neq \emptyset$
(4) a partitioning $A_1 \dot\cup \ldots \dot\cup A_m = \Sigma^k$ with $A_i \neq \emptyset$

This is a lot of input and indeed, the only input that is obvious is the training sample s. What does the additional input mean and how do we derive it?

Now, if we knew s to be a sample produced by a LDP (X_t) of dimension r with distribution P, then in above input we needed to choose $m = r$ and l, k, $(B_j)_j$ and $(A_i)_i$ such that the partitionings are indicative and characteristic events, that is result in a nonsingluar matrix $V = \mathsf{P}(B_j A_i)_{ij}$. The problem is that we do not have access to (X_t) and hence do not know r and P. We refer to these issues as determining the right or justifiable model dimension and as choosing good indicative and characteristic events. For now assume that we have made our choices.

Given the input of above, the basic OOM learning algorithm computes the target OOM $\tilde{\mathcal{A}} = (\mathbb{R}^m, (\tilde{\tau_a})_{a\in\Sigma}, \tilde{w_0})$ as

$$(\tilde{w_0})_i := \tilde{\mathsf{P}}_s(A_i) \tag{5.1}$$

$$\tilde{\tau_a} := \tilde{W_a}\tilde{V}^{-1} \tag{5.2}$$

where

$$\tilde{V} = (\tilde{V})_{ij} := \tilde{\mathsf{P}}_s(B_j A_i) \tag{5.3}$$

$$\forall a \in \Sigma \ : \ \tilde{W_a} = (\tilde{W_a})_{ij} := \tilde{\mathsf{P}}_s(B_j a A_i) \tag{5.4}$$

and

$$\tilde{\mathsf{P}}_s(A_i) \ := \ \frac{\#\{t \in \{1,\ldots,(|s|-k+1)\} \ : \ s[t:t+k] \in A_i\}}{|s|-k+1} \tag{5.5}$$

$$\tilde{\mathsf{P}}_s(B_j A_i) \ := \ \frac{\#\{t \in \{1,\ldots,(|s|-l-k+1)\} \ : \ s[t:t+l+k] \in B_j A_i\}}{|s|-l-k+1} \tag{5.6}$$

$$\tilde{\mathsf{P}}_s(B_j a A_i) \ := \ \frac{\#\{t \in \{1,\ldots,(|s|-l-k)\} \ : \ s[t:t+k+l+1] \in B_j a A_i\}}{|s|-l-k} \tag{5.7}$$

Here, $a \in \Sigma$ is an arbitrary single symbol and the terms $\tilde{\mathsf{P}}_s(A_i)$, $\tilde{\mathsf{P}}_s(B_j A_i)$ and $\tilde{\mathsf{P}}_s(B_j a A_i)$ are the empirical frequencies of occurrence of the events A_i, B_j and $B_j a A_i$ in the sample s. The derivation of these formula is straightforward and essentially builds on proposition 4.2 for interpretable OOMs. The details may be found in [**Jae98**], [**Jae00**].

When looking at the formulas just presented making up the basic OOM learning algorithm one can only wonder about it's simplicity. The substantial work to be done (besides matrix multiplications and inversions) consists of counting the occurrences of certain well defined events within the string s, that is a string matching and counting task. Hence, the algorithm not only is it simple, but also constructive and the computational work done is modest. In fact, in [**Jae00**] the run-time for the algorithm is argued to be $\mathcal{O}(n + |\Sigma|m^3)$. What follows is a discussion of some issues related to the run-time and space complexities of the basic OOM learning algorithm.

The run-time needed for a naive single sweep counting procedure is linear in the length of s only if we are allowed lookups of the form $s[t : t + l + k + 1] \in B_j a A_i$ in *constant* time. This can trivially be done by storing an array of size $|\Sigma^{l+k+1}|$ which then contains the index i of the characteristic event A_i, the index pair $(j, i)_1$ for matching $B_j A_i$ and the index pair $(j, i)_2$ for matching $B_j a A_i$. However, this results in storage requirements exponential in $l + k$ and hence quickly becomes prohibitive.

I will present an advanced method based on suffix trees later. This method retains the linear run-time for the counting procedure while at the same time consumes $\mathcal{O}(n \log m)$ memory.

Generally, when giving run-time complexities one must also give space complexities. This is the case since with unlimited memory (accessible in constant time), every problem can be solved in $\mathcal{O}(n)$ by simply storing precomputed solutions for every conceivable input of length n and using the input as an index into the solution array. The "processing" to be done then amounts to reading the input and looking up the solution. Obviously, an algorithm that consumes exponential memory in the input length is just cheating.

Also note, that the run-time thus obtained does only include the processing that has to be done *after* suitable indicative and characteristic events have been chosen. As the reader may guess, the hard part is in finding good indicative and characteristic events and likely considerable resources will be consumed in solving this "preprocessing step" before the basic learning algorithm can be applied.

The situation can be compared with training feedforward neural networks. Here, one usually measures the complexity of training the network based on the amount of processing done only after choosing a particular network topology. Often, this amounts to the run-time and memory consumption of backpropagation. The work done in choosing the "right" topology is booked differently or not even discussed.

With training OOMs, the situation is comparable. As we have seen, the work done after choosing particular indicative and characteristic events amount to a time-complexity of $\mathcal{O}(n + |\Sigma| m^3)$ and a space complexity of $\mathcal{O}(n \log m + |\Sigma| m^2)$ (will be shown). The work done in finding good indicative and characteristic events is a different story.

5.4. The Asymptotic Learning Theorem

This section paraphrases the discussion of the basic OOM learning algorithm given in [**Jae00**] with special emphasis to the aspect of "asymptotical correctness". It differs in a more technical formulation to fit into this thesis and by folding in the notion of ergodicity, which is discussed in detail in section 8 and was already motivated in 5.1. I start by presenting

LEMMA 5.1 (adapted from [**Jae00**]). Let (X_t) be a stationary ergodic LDP of finite dimension with distribution P taking values in a finite alphabet Σ. Let $s \in \Sigma^\infty$ a realization of (X_t), $A_i \subset \Sigma^k$, $B_j \subset \Sigma^l$ and $\tilde{\mathsf{P}}_{s[1:n]}(A_i)$ and $\tilde{\mathsf{P}}_{s[1:n]}(B_j)$ the frequencies of A_i and B_j in the initial string $s[1:n] \in \Sigma^n$ of the realization s. Then $\mathsf{P} - a.s.$

$$(1) \quad \tilde{\mathsf{P}}_{s[1:n]}(A_i) \overset{n\to\infty}{\longrightarrow} \mathsf{P}(A_i)$$
$$(2) \quad \tilde{\mathsf{P}}_{s[1:n]}(B_j A_i) \overset{n\to\infty}{\longrightarrow} \mathsf{P}(B_j A_i)$$
$$(3) \quad \forall a \in \Sigma \ : \ \tilde{\mathsf{P}}_{s[1:n]}(B_j a A_i) \overset{n\to\infty}{\longrightarrow} \mathsf{P}(B_j a A_i)$$

PROOF. Since (X_t) is stationary ergodic by assumption, we are allowed by the ergodic theorem to exchange the estimator of the ensemble average $\frac{1}{|\Sigma^n|} \sum_{s_n \in \Sigma^n} \tilde{\mathsf{P}}_{s_n}(A_i)$ with a time average limit $\lim_{n\to\infty} \tilde{\mathsf{P}}_{s[1:n]}(A_i)$ for almost every realization s. □

Based on the lemma just given, we now can transport the convergence result through the other stations of the basic learning algorithm. In particular, the frequencies of events A_i, $B_j A_i$ and $B_j a A_i$ in the sample make up the matrices $\tilde{V}$ and $\tilde{W}_a$. Hence,

COROLLARY 5.1 (adapted from [**Jae00**]). Assume the situation as in lemma 5.1. Let $V = (\mathsf{P}(B_j A_i))$ and $W_a = (\mathsf{P}(B_j a A_i))$. Then $\mathsf{P} - a.s.$

$$||V - \tilde{V}|| \overset{n\to\infty}{\longrightarrow} 0$$
$$\forall a \in \Sigma \ : \ ||W_a - \tilde{W}_a|| \overset{n\to\infty}{\longrightarrow} 0$$

where $||\cdot||$ denotes a matrix norm (e.g. 2-norm).

PROOF. From the previous lemma we see that the convergence is P-a.s. in all respective matrix elements. This implies the P-a.s. convergence in matrix norm. □

The last station in the basic learning algorithm uses the estimates $\tilde{V}$ and $\tilde{W}_a$ to compute estimates for the operators $\tilde{\tau_a}$. Consequently, convergence is further transported which is summarized in the following

THEOREM 5.1 (adapted from [**Jae00**]). Let (X_t) be a stationary ergodic LDP of dimension m on a finite alphabet Σ and $s \in \Sigma^\infty$ a single realization of (X_t). Assume $(A_i)_i$ and $(B_j)_j$ are characteristic and indicative events of (X_t) and let $\mathcal{A}$ denote the concrete OOM of (X_t) which is interpretable with respect to $(A_i)_i$.

Then for the sequence $\tilde{\mathcal{A}}_n(s)$ of OOMs estimated from the initial strings $s[1:n]$ by applying the basic learning algorithm using $(A_i)_i$ and $(B_j)_j$ it holds

$$\tilde{\mathcal{A}}_n(s) \overset{n\to\infty}{\longrightarrow} \tilde{\mathcal{A}} \quad \mathsf{P}-a.s.$$

in some matrix norm.

PROOF. Since $V \overset{n\to\infty}{\longrightarrow} \tilde{V}$ and $W_a \overset{n\to\infty}{\longrightarrow} \tilde{W}_a$ ($\mathsf{P}-a.s.$) when the basic learning algorithm is deployed, it follows that

$$\tilde{\tau_a} = \tilde{W}_a \tilde{V}^{-1} \overset{n\to\infty}{\longrightarrow} W_a V^{-1} = \tau_a \quad \mathsf{P}-a.s.$$

□

5.5. Numerical Error Analysis of OOM Learning

The second phase of OOM learning is a merely mechanical computation that consist of solving linear systems

$$V^T\ (\tau_x)_i\ =\ (W_x^T)_i \quad i \in \{1,\ldots,m\},\ x \in \Sigma \tag{5.8}$$

which yields the operator estimates $\tau_x,\ x \in \Sigma$. Here, $(\tau_x)_i$ denotes the i-th column vector of the matrix τ_x.

The statistical fluctuations in the counting matrices V and W due to finite sample size give rise to estimation errors in the operators τ_x.

An ill-conditioned matrix V is especially bad, since statistical errors will then be greatly magnified.

The condition of a matrix V with respect to matrix norm $\|\cdot\|$ is given by

$$\mathrm{cond}(V) = \|V\| \cdot \|V^{-1}\| \tag{5.9}$$

Note that always $\mathrm{cond}(V) \geq 1$ and $\mathrm{cond}(V^T) = \mathrm{cond}(V)$. Often, the norm chosen is the euclidian norm, also called Frobenius-norm:

$$\mathrm{norm}_F(V)\ =\ ||V||_F\ =\ \sqrt{\sum_{i,j} |(V)_{ij}|^2} \tag{5.10}$$

in which case the condition of V is given by the relation of the largest and the smallest singular value of V. This is also the condition we will use throughout this text.

As already stated, solving the linear systems from equation 5.8 with additive disturbances $\Delta(V_{ij})$ and $\Delta(W_{ij})$ will result in variations in the solutions τ_x.

Here we use the notation $\Delta(V_{ij})$ to denote the matrix which contains the disturbances as entries that will get added to the undisturbed entries of (V_{ij}).

The relative errors in τ_x with respect to variations in the right-hand sides $(W_x^T)_i$ is then given by

$$\frac{\|\Delta(\tau_x)_i\|}{\|(\tau_x)_i\|}\ =\ \mathrm{cond}(V^T)\ \frac{\|\Delta(W_x)_i)\|}{\|(W_x)_i\|} \tag{5.11}$$

whereas the relative errors in τ_x with respect to variations in the matrix (V^T) is given by

$$\frac{\|\Delta(\tau_x)_i\|}{\|(\tau_x)_i\|}\ =\ \frac{\mathrm{cond}(V^T)}{1-\mathrm{cond}(V^T)\frac{\|\Delta(V^T)_i\|}{\|(V^T)_i\|}}\ \cdot\ \frac{\|\Delta(V^T)_i\|}{\|(V^T)_i\|} \tag{5.12}$$

A derivation of these formulas may be found in [**Sto79**], p152. The figure 5.2 shows the dependency of the relative error in the estimated OOM operators $\frac{\|\Delta(\tau_x)_i\|}{\|(\tau_x)_i\|}$ on the V matrix condition $\mathrm{cond}(V^T)$ and the statistical error or the relative error in V given by $\frac{\|\Delta(V^T)_i\|}{\|(V^T)_i\|}$.

Now, we may speculate whether the *minimization of condition* over all possible partitionings (choices of indicative and characteristic events) is a good learning strategy. Indeed, this was one line of thoughts when starting this thesis. However, it quickly became apparent that a blind minimization of V-matrix condition does not work. The minimization of the condition of the counting matrix V is trivial when no other restrictions are imposed on the complexity or capacity of the chosen partitioning.

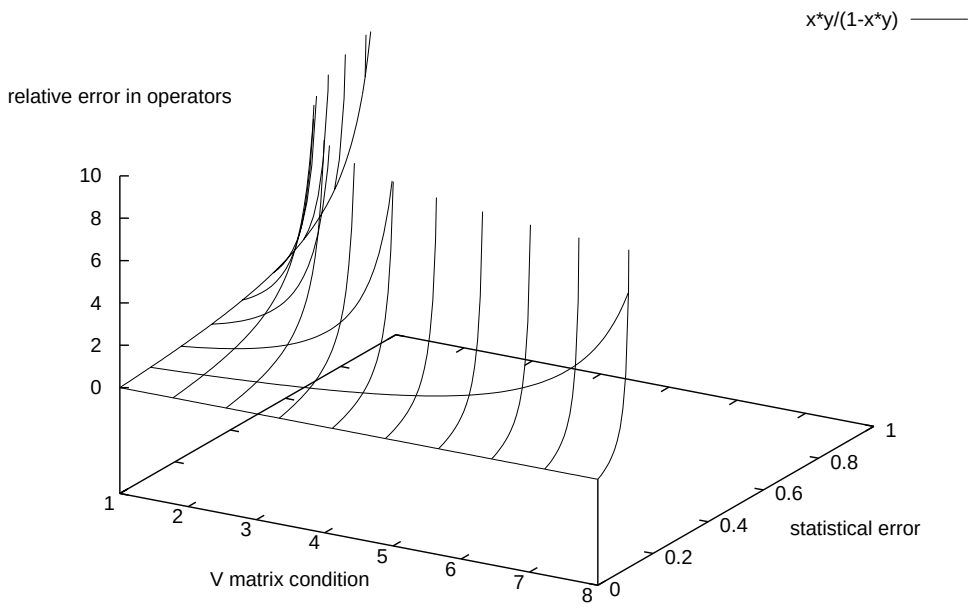

FIGURE 5.2. Influence of statistical errors in V and condition of V on estimation error in operators τ_x.

To show this, I will now give a constructive method of choosing characteristic and indicative events such that the V-matrix condition is (near) 1 (also see lemma 15.1).

This can be seen as follows. Suppose $s \in \Sigma^n$ is the given training sample and we want to estimate a m dimensional OOM. Let $s[:i]$ denote the prefix of s consisting of all characters of s up to the i-th character and let $s[i:]$ denote the suffix of s consisting of all characters of s starting from the i-th character.

Choose two arbitrary vectors $\eta^{1,2} \in \{1,\ldots,m\}^n$ and interpret $\eta^1_i = l_i$ as that the prefix $s[:i]$ is put into the indicative event l_i. Likewise interpret $\eta^2_i = k_i$ as that the suffix $s[i:]$ is put into the characteristic event k_i.

Let $\$ \notin \Sigma$ and define a partitioning $\pi_{indicative}$ of $(\Sigma \cup \$)^n$ into indicative events by specifying the indices of partitions that prefixes $\ldots\$s[:i]$ will get assigned to with

$$\pi_{indicative}(\$\ldots\$s[:i]) = l_i \iff \eta^1_i = l_i$$

and a similar partitioning $\pi_{characteristic}$ for characteristic events by

$$\pi_{characteristic}(s[i:]\$\ldots\$) = k_i \iff \eta^2_i = k_i$$

Then, by construction, these partitionings will give rise to $\eta^{1,2}$ when applied to $\ldots\$s\$\ldots$. Obviously, the corresponding counting matrix V can be directly computed from $\eta^{1,2}$ by

$$(V_{ij}) = \#\{r \in \{1,\ldots,n\} : \eta^1_r = i \wedge \eta^2_{r+1} = j\}$$

All this is a complicated way of stating that given complex enough partitionings, we can produce any counting matrix V with integer elements that sum up to the sample length n. If n is a multiple of m we thus can produce a perfect diagonal matrix V with entries n/m. This matrix has condition 1. If n is no multiple of m, some entries will have to deviate from n/m, but still the condition can be made ≈ 1.

But even if we restrict ourselves to partitionings of limited complexity, the minimization of the V matrix condition under the constraint of limited complexity still does not seem to be sufficient in general. On the other hand, a very high condition number (e.g. $>> 10 - 20$) often quickly results in bad and useless models in practice. This will be further discussed in later sections on the basis of experimental results and in the context of statistical learning theory.

In summary, the insights from numerical error analysis of OOM learning are important but likely will not tell the whole story. This is because learning in practice happens in the setting of *finite* sample data - a situation which cannot be fully analyzed using asymptotic theory.

Part 2

Information Theory and OOMs

The modern theory of information and communication is to a great extent based on the notion of information sources and the concept of entropy.

> "An *information source* or source is a mathematical model for a physical entity that produces a succession of symbols called 'outputs' in a random manner." [**Gra90**]

Information sources are nothing else than stochastic processes. The difference is mainly the perspective and interpretation. Since OOMs are perfect stochastic processes, it seems reasonable to expect some progress from applying information theory to our main theme, learning OOMs from finite data. In particular, an important aim of this part is to customize tools from information theory, e.g. to measure the quality of an estimated model.

I will only briefly sketch the notions and results from information theory as far as necessary in applying them to OOM learning. A comprehensive introduction to information theory may be found in [**CT91**].

CHAPTER 6

Information Measures

There are many ways to introduce entropy, mutual information and the other information measures, since there are many simple algebraic relations between them. One way is to introduce *divergence* as the single primal notion to build everything else on.

6.1. Divergence

> "The divergence plays a basic role in the family of information measures; all of the information measures that we will encounter - entropy, relative entropy, mutual information, and the conditional forms of these information measures - can be expressed as divergence." [**Gra90**], p23

Moreover, divergence - once defined over arbitrary probability measures - allows us to define all information measures for arbitrary valued random variables as well.

We first need a

DEFINITION 6.1. A measure λ on some measurable space $(E, \mathcal{E})$ is *absolutely continuous* with respect to another measure μ on $(E, \mathcal{E})$ *absolute continuous*

$$\mu \gg \lambda$$

if

$$\mu(E) = 0 \Rightarrow \lambda(E) = 0 \quad \forall E \in \mathcal{E}$$

DEFINITION 6.2 (Divergence, from [**Gra90**], p77). Given probability measures p and q on some measurable space $(\Omega, \mathcal{F})$, then the *divergence* of p with respect to q is given by *divergence*

$$D(p||q) = \begin{cases} \sup_Q \sum_{E \in Q} p(E) \log \frac{p(E)}{q(E)} & \text{if } q \gg p, \\ \infty & \text{else.} \end{cases}$$

where the supremum is over all finite measurable partitions Q of Ω.

Note, a finite measurable partition Q is given by $Q \subseteq \mathcal{F}$ where $|Q| < \infty$ and $E_1, E_2 \in Q \Rightarrow E_1 \cap E_2 = \emptyset$. Further, the definition is fully general, since we did not make any restrictions on the measurable space or the probability measures.

As common in information theory, log shall denote the logarithm to the base 2. Further, $0 \cdot \log \frac{0}{0} = 0$ since $0 \cdot \log 0 = 0$ as $\lim_{x \to 0} x \log x = 0$ and $p \log p/0 = \infty$ as $\lim_{x \to 0} p \log p/x = \infty$.

Formally, divergence is a functional of two probability measures. Looking at the definition, one may also recognize the general form of an expectation

$$D(p||q) = \sup_Q \sum_{E \in Q} p(E) \log \frac{p(E)}{q(E)} = \sup_Q \mathsf{E}_{p|Q} \log \frac{p(E)}{q(E)} \tag{6.1}$$

where $\mathsf{E}_{p|Q}$ denotes the expectation with respect to the measure p on the finite partition Q.

Before we present other notions, let us prove the following theorem, also known as the *divergence inequality*.

THEOREM 6.1. Given probability measures p and q like above, then

$$D(p||q) \;\geq\; 0$$

with equality if and only if $p = q$.

PROOF. The proof directly follows from the following the lemma and the fact that two measures p and q taking identical values on all finite partitions Q of Ω are identical. This is the case since $Q = A \cup \complement A$ for all $A \subset \Omega$ are finite partitions, and measures taking identical values on all $A \subset \Omega$ are trivially identical. □

LEMMA 6.1. Given two countable or finite sets of non-negative reals $\{p_i\}$ and $\{q_i\}$ such that $\sum p_i = \sum q_i = 1$, then

$$\sum p_i \log p_i/q_i \geq 0$$

with equality if and only if $p_i = q_i$ for all i.

PROOF. Since $\log(x) \leq x - 1$ with equality iff $x = 1$ it follows

$$\sum p_i \log p_i/q_i \leq \sum p_i(q_i/p_i - 1) = \sum q_i - \sum p_i = 0$$

with equality iff $q_i/p_i = 1$ for all i. □

The divergence inequality justifies the use of divergence as some kind of distance measure of probability measures. It is not a distance in the strict sense, since it is not symmetric and does not satisfy the triangle inequality. There are other functionals of probability measures intended for measuring proximity between probability measures. The functional I present next is one of them and - it is a true metric.

6.2. Hellinger Distance

The Kakutani-Hellinger distance or short Hellinger distance ([**Shi95**]) is, like divergence, a measure of proximity of two probability measures.

Hellinger distance

DEFINITION 6.3 (Hellinger Distance). Given probability measures p and q on some measurable space $(\Omega, \mathcal{F})$, then the *Hellinger distance* between p and q is given by

$$D^2_{\mathsf{HL}}(p \,||\, q) \;=\; \sup_Q \sum_{E \in Q} \left(\sqrt{p(E)} - \sqrt{q(E)}\right)^2$$

where the supremum is over all finite measurable partitions Q of Ω.

Unlike divergence, the Hellinger distance is a *metric* on the set of probability measures ([**Shi95**], 9, Lemma 3.2, p364). That is, it is symmetric in it's arguments and the triangle inequality holds. Later, we will use it for qualitative comparison with relative entropy.

6.3. Entropies of Random Variables

When divergence is applied to the probability distributions of random variables it becomes

relative entropy

DEFINITION 6.4 (Relative entropy). Let X and Y be random variables with distributions $p(X)$ and $p(Y)$. Then the *relative entropy* of X with respect to Y is given by

$$H_{X||Y}(X, Y) \;=\; D(p(X) \,||\, p(Y))$$

Relative entropy is also know as Kullback-Leibler divergence, Kullback-Leibler entropy, Kullback-Leibler distance and Kullback-Leibler information number. In this thesis, I will stick to the term relative entropy.

As we already mentioned, based on the notion of divergence we now can define all other information measures straight forward.

DEFINITION 6.5 (Entropy). Let X be a random variable with distribution $p(X)$. Let $p(X)p(X)$ denote the usual product measure and $p(X,X)(E_1,E_2) = p(X)(E_1)$ if $E_1 = E_2$ and 0 otherwise denote the diagonal measure on the product space. Then the *entropy* of X is given by *entropy*

$$\begin{aligned} H(X) &:= D(p(X,X) \mid\mid p(X)p(X)) \\ &= \sum_{x_1 \in \mathcal{X}} \sum_{x_2 \in \mathcal{X}} p(x_1,x_2) \log \frac{p(x_1,x_2)}{p(x_1)p(x_2)} \\ &= \sum_{x \in \mathcal{X}} p(x) \log p(x)^{-1} \\ &= -\mathsf{E}_{p(X)} \log p(X) \end{aligned}$$

DEFINITION 6.6 (Mutual Information, Joint Entropy, Conditional Entropy). Let X and Y be random variables with joint distribution $p(X,Y)$, conditional distribution $p(Y \mid X)$ and marginal distributions $p(X)$ and $p(Y)$. Then the *mutual information* between X and Y is the relative entropy between the joint distribution and the product distribution *mutual information* *joint entropy* *conditional entropy*

$$\begin{aligned} I(X;Y) &:= D(p(X,Y) \mid\mid p(X)p(Y)) \\ &= \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} \\ &= \mathsf{E}_{p(X,Y)} \log \frac{p(X,Y)}{p(X)p(Y)} \end{aligned}$$

the *joint entropy* between X and Y is

$$\begin{aligned} H(X,Y) &:= D(p(X,Y) \mid\mid p^2(X,Y)) \\ &= \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x,y) \log p(x,y)^{-1} \\ &= -\mathsf{E}_{p(X,Y)} \log p(X,Y) \end{aligned}$$

and the *conditional entropy* of Y given X is given by

$$\begin{aligned} H(Y \mid X) &:= D(p(X,Y) \mid\mid p^2(Y \mid X)) \\ &= \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x,y) \log p(y \mid x)^{-1} \\ &= -\mathsf{E}_{p(X,Y)} \log p(Y \mid X) \end{aligned}$$

A number of mechanical calculations the reader will find in any comprehensive text book on information theory (e.g. [**CT91**]) show that

LEMMA 6.2. Given two random variables X and Y. Then

(1) $I(X;X) = H(X) \geq 0$,
(2) $I(X;Y) \geq 0$ with equality iff X and Y are stochastically independent
(3) $I(X;Y) \leq H(X)$ and $I(X;Y) \leq H(Y)$
(4) $H(Y \mid X) \leq H(Y)$
(5) $I(X;Y) = H(X) + H(Y) - H(X,Y)$
(6) $I(X;Y) = I(Y;X) = H(X) - H(X \mid Y) = H(Y) - H(Y \mid X)$

(7) $\max(H(X), H(Y)) \leq H(X,Y) \leq H(X) + H(Y)$
(8) If X is finitely valued with values in $\mathcal{X}$, then $H(X) \leq \log |\mathcal{X}|$

6.4. Mutual Information of Matrices

Later, we will encounter matrices that store probabilities of joint events $B_j A_i$ in their entries. Also, it will be of interest to analyze the mutual information between random variables X and Y defined through those events A_i and B_j.

For convenience reasons, I introduce the following shortcut to refer to the mutual information of pairs of finitely valued random variables which joint probabilities are stored in matrices. Generally, this allows to speak of the mutual information of matrices with all positive entries that sum up to 1.

Definition 6.7. Given a matrix $V \in \mathbb{R}^{n \times m}$ with $V_{ij} \geq 0$ and $\sum_{i,j} V_{ij} = 1$. Then the mutual information of V is given by

$$I(V) \ := \ \sum_{i,j} V_{ij} \log \frac{V_{ij}}{\sum_k V_{kj} \cdot \sum_k V_{ik}}$$

This merely is a direct transcription of mutual information as introduced in definition 6.6. Trivially, we can always normalize a matrix with positive entries such that the entries sum up to 1.

CHAPTER 7

Information Rates

The last section introduced the basic information measures on single random variables or pairs of random variables. Since our main concern is stochastic processes, that is sequences of random variables, a natural question to ask is how the joint or conditional entropy of a sequence of random variables develops as the sequence gets longer and longer. This immediately leads to the notion of information rates, which subsequently will be our major tool for comparing OOMs, measuring the distance between OOMs and also measuring quality of model estimation.

7.1. Entropy Rate

Entropy rates can be defined in two ways. First, entropy rate can be defined as the normalized limit of the joint entropy of finite initial sequences of the random variables giving rise to the process

DEFINITION 7.1. The *entropy rate* of a stochastic process (X_t) is given by *entropy rate*

$$h((X_t)) = \lim_{n\to\infty} \frac{1}{n} H(X_1, \ldots, X_n)$$

when the limit exists.

The second way of defining entropy rate is by the limit of the conditional entropy of initial sequences of the random variables conditioning the next step random variable

DEFINITION 7.2. The *conditional entropy rate* of a stochastic process (X_t) is given by *conditional entropy rate*

$$h'((X_t)) = \lim_{n\to\infty} H(X_n \mid X_{n-1}, \ldots, X_1)$$

when the limit exists.

Luckily, the following theorem shows that at least in the case of stationary processes we do not have to commit to one over the other definition.

THEOREM 7.1 (from [**CT91**]). For a stationary stochastic process (X_t), the limits in definitions 7.1 and 7.2 exist and are equal

$$h((X_t)) = h'((X_t))$$

PROOF. Since conditioning over more random variables reduces conditional entropy (see lemma 6.2(4)) and (X_t) is stationary, it follows that

$$H(X_{n+1} \mid X_n, \ldots, X_1) \leq H(X_{n+1} \mid X_n, \ldots, X_2) = H(X_n \mid X_{n-1}, \ldots, X_1)$$

is a decreasing sequence of non-negative reals and hence converges to a limit $h'((X_t))$. Further, by the chain rule for the joint entropy of random variables

$$\frac{H(X_1, \ldots, H_n)}{n} = \frac{1}{n} \sum_{i=1}^{n} H(X_i \mid X_{i-1}, \ldots, X_1)$$

The Cesaro mean tells us, that given $a_n \to a$ and $b_n = \frac{1}{n}\sum_{i=1}^{n} a_i$ results in $b_n \to a$. Hence,

$$\begin{aligned} h((X_t)) &= \lim_{n\to\infty} \frac{H(X_1,\ldots,X_n)}{n} \\ &= \lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} H(X_i \mid X_{i-1},\ldots,X_1) \\ &= \lim_{n\to\infty} H(X_n \mid X_{n-1},\ldots,X_1) = h'((X_t)) \end{aligned}$$

□

Note, that the above theorem holds for any stationary stochastic process. The importance of the entropy rate is given by the fact that the entropy rate of a stationary ergodic process gives the average information contained per symbol. Precisely, the expected description length of a string of length n generated by a stationary ergodic process (X_t) is given by $n \cdot h((X_t))$. We will proof this in a later section.

7.2. Entropy Rates of OOMs

We now discuss some experimental results for concrete OOMs.

7.2.1. Experiments. The figures 7.1 and 7.2 show experimental results for approximation of the entropy rates of two different OOMs computed as in definition 7.1 for finite inital sequences up to a length of 10 and 15 respectively.

Precisely, the calculations were done as follows. For each "sample length" n in the ranges $1-15$ (or $1-10$) the value plotted was computed as

$$y(n) \;=\; 1/n \cdot H(X_1,\ldots,X_n) \;=\; 1/n \sum_{w\in\Sigma^n} -p(w)\log(p(w)) \tag{7.1}$$

The two OOMs used are the following.

The "Probability Clock". A 3-dimensional OOM on the alphabet $\Sigma = \{a, b\}$, which is not describable as a (finite state) Hidden Markov Model, defined by

$$\mathcal{A} = (\mathbb{R}^3, (\tau_x)_{x\in\{a,b\}}, w_0) \tag{7.2}$$

where

$$\tau_a = 0.5 \begin{pmatrix} 1 & 0 & 0 \\ 0 & c & s \\ 0 & -s & c \end{pmatrix},$$

$$\tau_b = \begin{pmatrix} .75\cdot.5 & .75(1-.5c+.5s) & .75(1-.5s-.5c) \\ 0 & 0 & 0 \\ .25\cdot.5 & .25(1-.5c+.5s) & .25(1-.5s-.5c) \end{pmatrix}$$

with $s = \sin(1)$, $c = \cos(1)$ and w_0 is the eigenvector of $\tau_a + \tau_b$ to the eigenvalue 1, approximately

$$w_0 = (0.787274, 0.077788, 0.134938)$$

A 3-dimensional OOM on the alphabet $\Sigma = \{a, b, c\}$, which is also a Hidden Markov Model, given by

$$\mathcal{A} = (\mathbb{R}^3, (\tau_x)_{x\in\{a,b,c\}}, w_0) \tag{7.3}$$

where

$$\tau_a = \begin{pmatrix} 0.09 & 0 & 0.05 \\ 0.01 & 0.45 & 0 \\ 0 & 0.05 & 0.45 \end{pmatrix}, \quad \tau_b = \begin{pmatrix} 0.18 & 0 & 0.04 \\ 0.02 & 0.045 & 0 \\ 0 & 0.005 & 0.36 \end{pmatrix},$$
$$\tau_c = \begin{pmatrix} 0.63 & 0 & 0.01 \\ 0.07 & 0.405 & 0 \\ 0 & 0.045 & 0.09 \end{pmatrix}$$

and

$$w_0 = (1/3, 1/3, 1/3)^T$$

7.2.2. Discussion. The approximations of entropy rates for the case when the models are started from an invariant starting distribution, which results in a stationary OOM, are monotonically decreasing and obviously already can be seen to converge for small initial sequence lengths (thick lines).

When the models are started from a non-invariant starting distribution and hence behave as non-stationary stochastic processes, the theorem proved above will not generally hold. However, at least in the plots shown we can see that also in these cases the non-stationary initial transient of the models does not prevent the entropy rate approximations from converging to the *same* limit, though not always monotonically (dashed lines). A plausible explanation could be that if the non-stationary OOM does have a geometrically fast decaying memory, any initial transients of the OOM will be quickly "forgotten" and after that, when a quasi-stationary regime is reached, the OOM behaves as if it was stationary. Thus, any finite contributions from the initial transients to the sums under the limit in the entropy rate formulas will ultimately vanish. The contributions of initial transients will be finite only if the memory decay is geometrically fast.

It would be interesting to further investigate entropy rates and specifically convergence criteria of entropy rates for non-stationary OOMs. In this context, it might be of value to analyze the rate of memory decay of certain OOMs in general.

7.3. Relative Entropy Rate

The entropy rate is an important characteristic of a stochastic process. However, it is possible for two stochastic processes to have identical entropy rates but still be completely different. What we want is a measure of similarity of stochastic processes. We already have seen the notions of divergence, which measures a distance between probability distributions and the analog of divergence for random variables, the relative entropy.

Unfortunately, computing the relative entropy of inital sequences of random variables does not make sense since this diverges in the sequence length. Hence, it seems reasonable to normalize the relative entropy by the sequence length to a relative entropy rate

relative entropy rate

DEFINITION 7.3 (from [**Gra90**]). Given two stochastic processes (X_t) and (Y_t) with distributions p_X and p_Y, the *relative entropy rate* of (X_t) with respect to (Y_t) is given by

$$\bar{h}_{p_X||p_Y}((X_t),(Y_t)) = \limsup_{n\to\infty} \frac{1}{n} H_{p_X||p_Y}((X_n,\ldots,X_1) \mid\mid (Y_n,\ldots,Y_1))$$

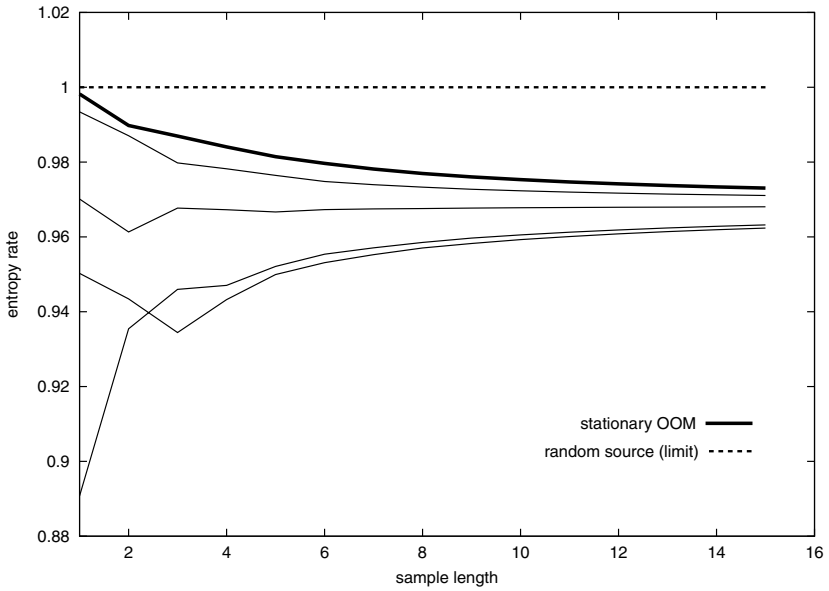

FIGURE 7.1. Finite-length approximations of entropy rate of the probability clock OOM. Thick line shows the entropy rate approximation for the stationary case, whereas other lines show entropy rate approximations for non-stationary cases.

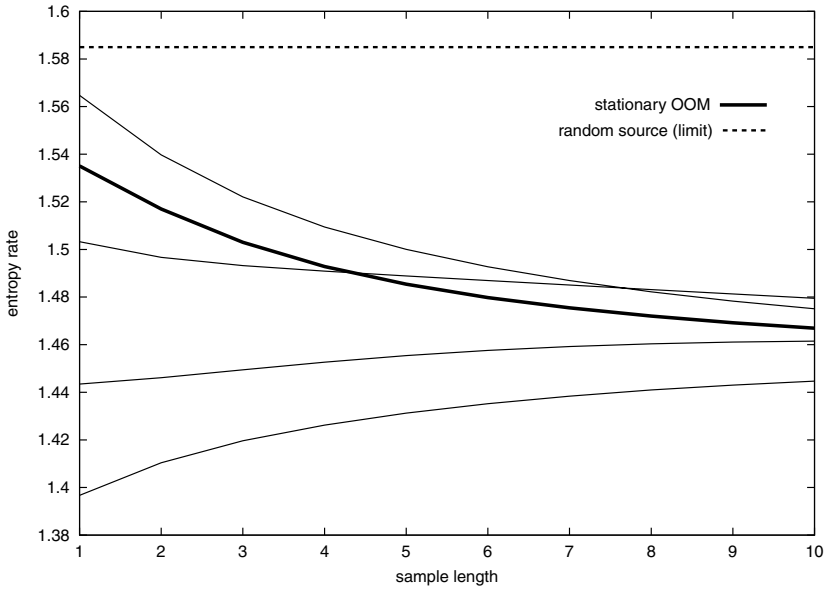

FIGURE 7.2. Finite-length approximation of entropy rate of a 3-dimensional HMM. Thick line shows the entropy rate approximation for the stationary case, whereas other lines show entropy rate approximations for non-stationary cases.

The relative entropy rate between two stochastic processes was only proven to be finite in some situations. If, for example, one of the processes is a k-order Markov process dominating the other, then the relative entropy rate between the two is finite [**Gra90**]. The question of finiteness of the relative entropy rate in the general case of arbitrary OOMs is unknown. However, there exists a positive result for Hidden Markov Models [**JR85**]. Thus, at least for OOMs that characterize stochastic processes which can also be described as HMMs, the relative entropy rate is finite.

7.4. Monte-Carlo Integration

This section introduces the method of Monte-Carlo integration (see [**Sin92**]) which is used for approximating sums and integrals.

I will show how to apply Monte-Carlo integration to the problem of approximating relative entropy rates. Then, in the next section, experimental results will illustrate the practical value of the method.

Instead of directly evaluating an integral, which often is infeasable, Monte-Carlo integration offers a generic transformation into an equivalent expected value computation problem.

Given a continous function $f : \mathbb{R} \to \mathbb{R}$ and an arbitrary probability density function u on $\mathbb{R}$ such that $\{x \in \mathbb{R} : u(x) = 0 \ \wedge \ f(x) \neq 0\}$ is countable, then

$$\int f(x)\,dx \ = \ \int \frac{f(x)}{u(x)} \cdot u(x)\,dx \ = \ \mathsf{E}_u\left(\frac{f(x)}{u(x)}\right) \tag{7.4}$$

The expectation $\mathsf{E}_u(\frac{f(x)}{u(x)})$ may be estimated by *sampling* u. Sampling u in this context means drawing a finite number of samples $x_1, \ldots, x_k$ according to the probability density u. Then (sloppy):

$$\mathsf{E}_u\left(\frac{f(x)}{u(x)}\right) \ \approx \ \frac{1}{k}\sum_{i=1}^{k}\frac{f(x_i)}{u(x_i)} \tag{7.5}$$

The method can be applied to the problem of approximating both the relative entropy rate and the Hellinger distance between stochastic processes, as is shown in this thesis in the following.

The divergence between probability measures p and q on a countable probability space Ω can be approximated by

$$\begin{aligned} D(p||q) = \sum_{\omega\in\Omega} p(\omega)\log\frac{p(\omega)}{q(\omega)} = \mathsf{E}_u\left(\frac{p(\omega)}{u(\omega)}\cdot\log\frac{p(\omega)}{q(\omega)}\right) \\ \overset{u=p}{=} \mathsf{E}_u \log\frac{p(\omega)}{q(\omega)} \approx \frac{1}{k}\sum_{i=1}^{k}\log\frac{p(\omega_i)}{q(\omega_i)} \end{aligned} \tag{7.6}$$

Note, that here we have sampled according to $u = p$, which makes sense if p is our reference distribution.

Similarily, the Hellinger distance can be approximated by

$$\begin{aligned} D^2_{\mathsf{HL}}(p\,||\,q) = \sum_{\omega\in\Omega}\left(\sqrt{p(\omega)} - \sqrt{q(\omega)}\right)^2 = \mathsf{E}_u\left(\frac{1}{u(\omega)}\cdot\left(\sqrt{p(\omega)} - \sqrt{q(\omega)}\right)^2\right) \\ \overset{u=p}{\approx} \frac{1}{k}\sum_{i=1}^{k}\left(\frac{1}{p(\omega)}\cdot\left(\sqrt{p(\omega)} - \sqrt{q(\omega)}\right)^2\right) \end{aligned} \tag{7.7}$$

The significance of all this for our purposes is, that it enables us to approximate the relative entropy rate between stochastic processes more precisely given finite computational resources.

This can be seen from the experimental results presented in the next section where the Monte-Carlo integration method as applied to information rate approximation is compared to a more direct method.

7.5. Relative Entropy and Hellinger Rates of OOMs

This section will discuss some experiments addressing the usefulness of relative entropy rate and Hellinger rate in measuring the distance between OOMs and evaluating the quality of estimated OOMs. The figures 7.3, 7.4, 7.5 and 7.6 show the obtained results.

7.5.1. Experiments. The experiments were done as follows. A given OOM, subsequently called the target OOM, was used to generate samples. Based on these samples, new OOMs were estimated. Then, the relative entropy rates between the target OOM and the estimated OOMs were analyzed.

One group of experiments studied the consequences of sample size. Here, models where estimated based on increasing length of training sample: 50k, 100k and 200k data. Results are given in figures 7.3 and 7.5.

The other group of experiments studied the influence of specific choices of indicative and characteristic events (model a,b and c) while the training sample length was constant (100k). Results are shown in figures 7.4 and 7.6.

Also, both groups of experiments were done for the relative entropy rate and the Hellinger rate.

The relative entropy rates were approximated using two different methods. The thick lines in the figures show approximations of the relative entropy rates using an "exact" method: for each "sample length" n in the range $1 - 12$ the value plotted was computed as

$$y(n) \;=\; 1/n \sum_{w \in \Sigma^n} p(w) \log(\frac{p(w)}{q(w)}) \tag{7.8}$$

where p and q are the probability distributions of the target OOM and the estimated OOM respectively.

Longer sample lengths quickly consume prohibitive many CPU cycles, since the number of finite sequences w growths exponentially in the sequence length. Therefore I developed a second approximation method which applies the technique of Monte-Carlo integration to the problem of relative entropy apporoximation.

The thinner and dashed lines give the relative entropy rate approximations by using Monte-Carlo integration. Here, 10^3 samples each of length n were drawn according to the distribution of the target OOM at each of the sample lengths n in the range of $1 - 60$ and the plotted values were computed as

$$y(n) \;=\; \frac{1}{10^3 n} \sum_{w \in S} \log(\frac{p(w)}{q(w)}) \tag{7.9}$$

where S is the sample of size 10^3 drawn from the target OOM.

The target OOM used was a 3-dimensional OOM on the alphabet $\Sigma = \{a, b, c\}$ as defined in equation 7.3.

7.5.2. Discussion. Several aspects seem worth mentioning. First, the figures 7.3 and 7.5 suggest that the relative entropy rate approximations converge. In other words, the limit that defines relative entropy rates exist. Note that the target OOM used in the experiments was equivalent to a HMM. These findings are in accordance with [**JR85**] where the existence of the relative entropy rates for Hidden Markov Models was proved.

It would be interesting however, to derive a similar result even for the case of arbitrary non-HMM Observable Operator Models. That is, a proof of existence of the relative entropy rate between arbitrary OOMs.

Second, if we compare the results given by the "exact" method for short sample length (thick lines) and the results obtained based on Monte-Carlo integration (dashed/thin lines) for greater length, we see that the Monte-Carlo method provides better results. This is the case since the relative entropy rate is monotonically increasing. Hence, the "exact" method underestimates the information rate or distance between the stochastic processes in the example given. Also, the computational resources consumed for approximating the relative entropy rate at sample length 60 based on 10k samples is much less than computing the relative entropy rate for all samples of length 12.

I therefore conclude that the Monte-Carlo method of approximating information rates introduced in this thesis is advantageous.

Third, another experimental result is that longer training data sizes indeed lead to better models both for the relative entropy rate and the Hellinger rate. Of course, nothing else was expected. Yet the model improvements seem to be non-proportional in the training data size. Doubling the training data size from 50k to 100k and 200k does not result in a proportional but limiting decrease of the distance of the so estimated models with respect to the target model in this experiment.

Intuitively, it seem plausbile that increasing the training data size further and further only results in diminishing improvements in the quality of the estimated model.

Forth, regarding the influence of particular choices of indicative and characteristic events (partitionings), in figures 7.4 and 7.6 we see that indeed partitionings are of considerable effect on the model quality independent of the training data size.

This strengthens the perspective that learning based on *finite* training data is fundamentally different from asymptotic learning from infinite training data. Only for the latter the asymptotic learning theorem for OOMs tells us that, modulo certain weak restrictions ("nonsingular V matrix"), the specific choice of indicative and characteristic events is irrelevant.

The experimental results presented here suggest that on the basis of finite training data the choice of partitioning can make all the difference between good and poor model estimates.

And fifth, from the figures it seems unclear if the Hellinger distance approximations converge. Thus, it is unclear if the limit defining the Hellinger rate exists. Also, I am not aware of any theoretical results in this direction. However, the results seem to be qualitatively similar to those obtained based on relative entropy rates when it comes to differentiating the models according to their quality or distance. And, the Hellinger rate seems to be smoother and more stable to approximate.

I conclude, that the Hellinger rate may advantagous to the relative entropy rate when it is only important to differentiate models on a qualitative but reliable level using as little computational resources as possible.

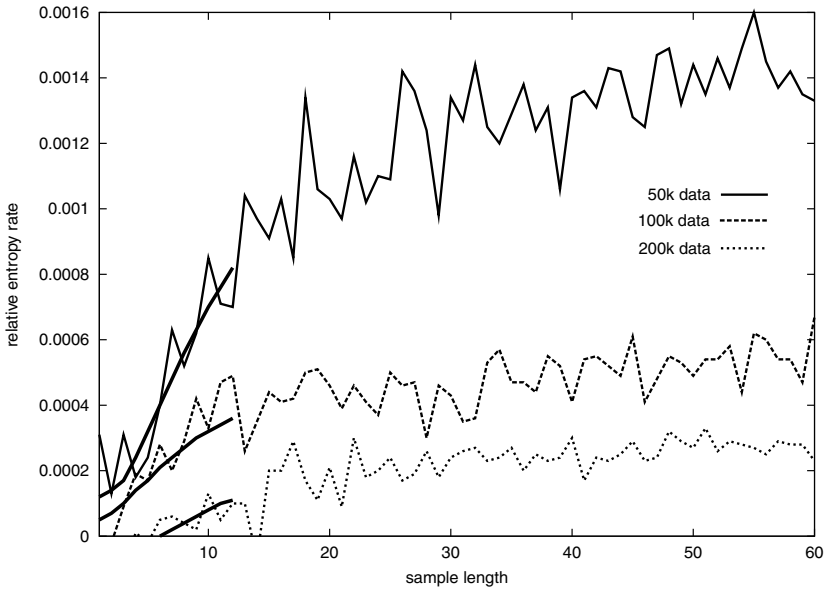

FIGURE 7.3. Relative entropy rate approximations between 3 models estimated from different training data sizes and the target OOM.

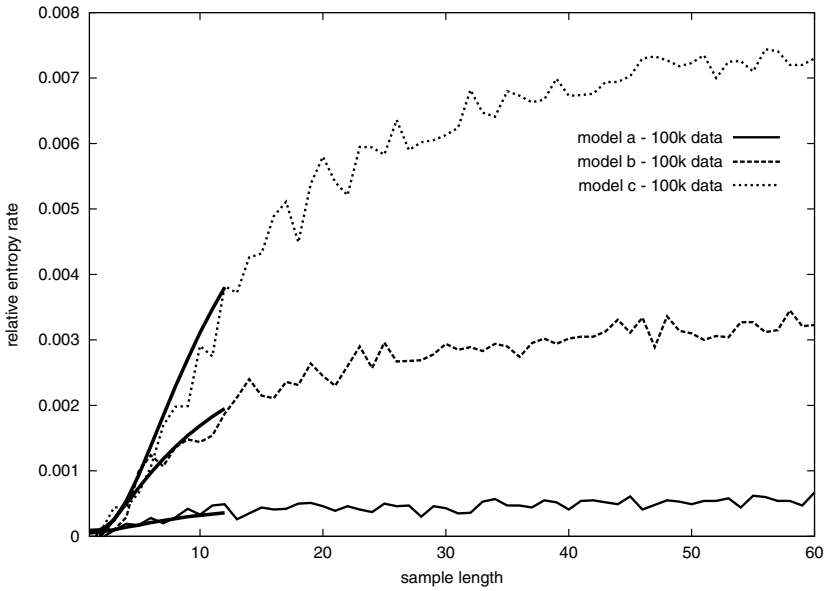

FIGURE 7.4. Relative entropy rate approximations between 3 models estimated based on different indicative/characteristic events and the target OOM.

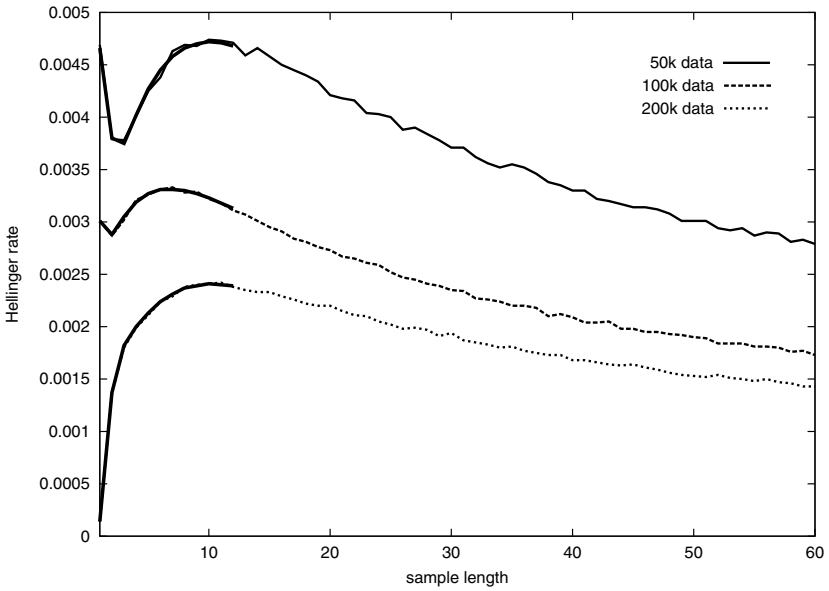

FIGURE 7.5. Hellinger rate approximations between 3 models estimated from different training data sizes and the target OOM.

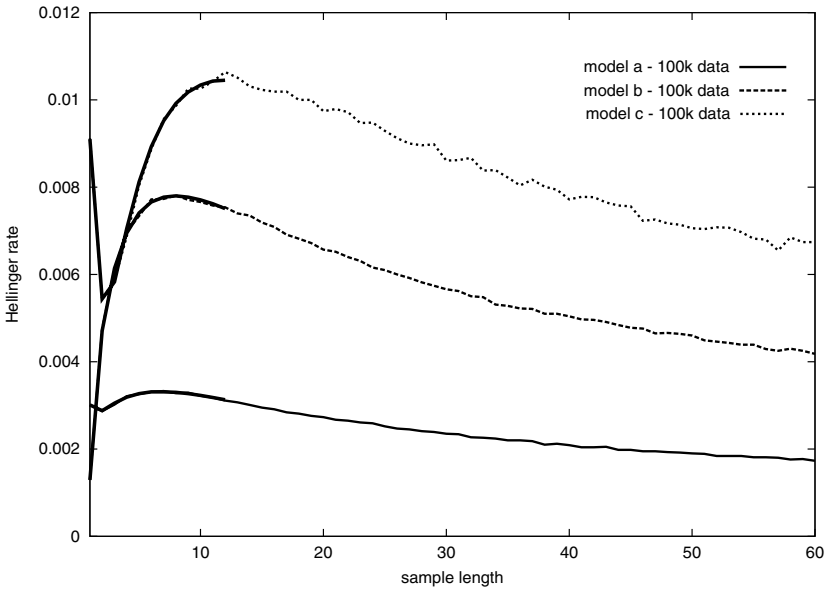

FIGURE 7.6. Hellinger rate approximations between 3 models estimated based on different indicative/characteristic events and the target OOM.

CHAPTER 8

Ergodicity

Ergodic theory is the study of measure preserving transformations on a probability space. Stationary stochastic processes can be studied as measure preserving transformations and vice versa. The standard definition of ergodicity of stochastic processes then is in terms of properties of the corresponding measure preserving transformation. Hence, to talk of ergodicity in the context of stochastic processes, one had to first introduce some ergodic theory in general, but a formal introduction of ergodic theory is clearly beyond the scope of this thesis. The reader may find such an introduction restricted to the perspective of stochastic processes in [**Shi95**].

Here, I will only motivate ergodicity of stochastic processes as far as it is needed to then (formally) state a number of interesting aspects which relate to information theory. These will then be applied to the problem of OOM estimation within the context of statistical learning theory.

8.1. Ergodic Processes

stationary stochastic process

A stochastic process is stationary if *all* its statistical properties are invariant with respect to time. Hence, for a stationary (in the strict sense) stochastic process $(X_t)_{t\in T}$ taking values in $\mathcal{X}$, functions of the random variables X_t making up the stochastic process are independent of time translations, e.g. n-th moments of a stationary process are time independent

$$\mathsf{E}^{(\mathsf{n})}(X_t) = \mathsf{E}^{(\mathsf{n})}(X_{t'}) \quad \forall t, t' \in T \tag{8.1}$$

ergodic stochastic process

A trivial example of a stationary stochastic process is a white-noise (iid) process. A stationary process is also *ergodic* if *all* its statistical properties can be inferred from almost any *single realization* or outcome of the process. In other words, almost every realization of a stationary ergodic process exhibits statistical properties that are characteristic of the whole process.

Ergodic stochastic processes are special stationary stochastic processes. The crucial difference between arbitrary stationary and ergodic stochastic processes is that for the latter the probability distribution has a special form which allows one to compute probabilities of events both by *ensemble averaging* and by *time averaging*.

In physics, such situations usually are paraphrased by stating: "we are allowed to exchange an ensemble average with a time average".

A stationary stochastic process where all statistical quantities computed as time averages are equal to the corresponding ensemble averages is called ergodic.

A time average is computed from averaging a desired function over a *single realization* of the stochastic process in time, whereas with ensemble averages the function is averaged (weighted according to probability) over all possible realizations, the ensemble.

For our previous example of n-th moments, the equality between ensemble and time averages can be formulated as follows.

$$\mathsf{E}^{(\mathsf{n})}(X_t) = \int_{x\in\mathcal{X}} x^n P(X_t = x)dx = \int_{t\in T} \mathbf{x}(t)^n dt \tag{8.2}$$

where $\mathbf{x} \in \mathcal{X}^T$ is a single realization of the stochastic process and $\mathbf{x}(t)$ denotes the value for the realization $\mathbf{x}$ at time t. For a discrete time ($T = \mathbb{N}$), real valued ($\mathcal{X} = \mathbb{R}$) stationary ergodic process (X_t) the above equality can be expressed

$$\mathsf{E}^{(\mathsf{n})}(X_t) = \int_{x\in\mathbb{R}} x^n P(X_t = x)dx = \lim_{s\to\infty} \frac{1}{2s} \int_{-s}^{+s} \mathbf{x}(t)dt \tag{8.3}$$

Since every property of a stochastic process (X_t) can be expressed as the expectation of a suitable defined sequence of random variables (ξ_t) defined on top of the stochastic process, the reader may look at the following theorem as a formalization of the introducing comments about ergodic processes.

THEOREM 8.1 (from [**Shi95**]). Let $(\xi_t)_{t\in\mathbb{N}}$ be a stationary (in the strict sense) ergodic sequence of random variables with finite expectation. Then

$$\frac{1}{n}\sum_{t=1}^{n} \xi_t(w) \to \mathsf{E}(\xi_1) \quad \text{P-almost surely}$$

A proof may be found in [**Shi95**]. Important ergodic processes are for example Markov chains with transition probabilities $p_{ij}^n > 0\ \forall i, j$ for some n [**Sin92**]. Also, many real-world processes which are stationary are also ergodic. A simple example of a stochastic process that is stationary, but not ergodic is a random constant process, where the outcome at time 0 is random, but determines the value the process takes for all $t > 0$. Since this example process is trivially a stationary LDP, I conclude that there are stationary non-ergodic LDPs.

In the non-ergodic situation, one can not infer any statistical property of the process by looking at a single realization of the process.

I find this point quite far reaching, when generalized to the problem of learning from data as such. My argument is this: **How could one expect to reliably estimate a model from a sample sequence when the underlying process is *not* ergodic?** When the process is not ergodic, it might not help to have a longer and longer sample. Even an infinite sample sequence will not grasp all process properties in general. Without additional knowledge about the process, learning in this setting does make little sense. In other words, **learning models from data produced by a non-ergodic or even non-stationary process seems to *require* apriori knowledge**. This apriori knowledge will sneak in by restricting the class of candidate probability distributions, a parametrized model class.

In the next sections, we will look at a number of important information theoretic properties of stationary ergodic processes.

We will restrict our formulations to discrete time, finite valued processes. In the next sections, I use $(X_t)_{t\in\mathbb{N}}$ to denote a discrete time stochastic process that takes values in a finite set Σ. Further, assume that (X_t) is *stationary ergodic* and specified by distribution P (on finite initial sequences for example). We use $X_1^n = (X_1, X_2, \ldots, X_n)$ to denotes a finite inital sequence of random variables that define (X_t).

8.2. Typical Sequences

Stochastic processes which are stationary and ergodic have a number of astonishing and far reaching properties. One of those is that the set of sequences they produce can be divided into two classes: the *typical set* and everything else ([**Wyn95**], [**CT91**]).

As it turns out, the set of typical sequences has these properties (*)

(1) it has probability near 1,
(2) it is not "too big",
(3) all sequences within the set have approximately equal probability and
(4) the probability in (3) is a function of the entropy rate of the process

Every property that is proved for a typical sequence will then hold true with high probability in general and for any sample that is long enough in particular.

The properties of typical sets just enumerated will now be presented formally and in greater detail.

typical set

DEFINITION 8.1 (from [**Wyn95**]). Let $\epsilon > 0$, $n \in \mathbb{N}$ and h the entropy rate of (X_t). Then the *typical set* of (X_t) is given by

$$T(n,\epsilon) \;=\; \{w \in \Sigma^n : |\frac{1}{n}\log\frac{1}{P(X_1^n = w)} - h| \leq \epsilon\} \tag{8.4}$$

In other words, the typical set of (X_t) to the parameters ϵ and n consists of all finite strings w of length n such that the probability of observing w is bounded by the inequality $2^{-n(h+\epsilon)} \leq P(w) \leq 2^{-n(h-\epsilon)}$ (property (3) in (*)).

The typical set $T(n,\epsilon)$ is roughly the same as the collection of substrings of length n in a sample of X_1^N where $N \approx 2^{n(h+\epsilon)}$ ([**Wyn95**]).

8.3. The Asymptotic Equipartition Theorem

The Asymptotic Equipartition Theorem (AEP) can be considered the analog of the weak law of large numbers in information theory.

THEOREM 8.2 (Asymptotic Equipartition Theorem, from [**Wyn95**]). For a stationary ergodic source

$$\forall\epsilon > 0 \;:\; \lim_{n\to\infty} P(T(n,\epsilon)) = 1$$

that is

$$\frac{1}{n}\log\frac{1}{P(X_1^n)} \to h \quad \text{in probability P}$$

Hence, finite sequences produced by the stochastic process can be found in the typical set with probability near one (property (1) in (*)). A proof of the theorem may be found [**Gal68**]. However, there is a stronger version of the theorem which I present in the next section and the reader may be more interested in the proof of that one.

Of course, without placing bounds on the size of typical sets, one may well object the usefulness of the AEP at all. Fortunately, the following property gives an upper (lower) bound on the size of typical sets (property (2) in (*))

PROPOSITION 8.1 (from [**Wyn95**]). $2^{n(h-\epsilon)} \leq |T(n,\epsilon)| \leq 2^{n(h+\epsilon)}$

PROOF. Extended from [**Wyn95**]: the right inequality follows directly from

$$1 \geq P(T(n,\epsilon)) = \sum_{w \in T} P(w) \geq 2^{-n(h+\epsilon)}|T(l,\epsilon)|$$

The left inequality follows from

$$1 - \epsilon < P(T(n,\epsilon)) = \sum_{w \in T} P(w) \leq 2^{-n(h-\epsilon)}|T(l,\epsilon)|$$

□

8.4. The Shannon-McMillan-Breiman Theorem

The Shannon-McMillan-Breiman theorem (SMB) is a stronger version of the AEP ([**Wyn95**]). It is the analog of the strong law of large numbers in information theory.

THEOREM 8.3 (from [**Wyn95**], [**CT91**]). For a stationary ergodic source

$$P(w : \frac{1}{n} \log \frac{1}{P(X_1^n(w))} \to h) = 1$$

that is

$$\frac{1}{n} \log \frac{1}{P(X_1^n)} \to h \quad \text{P-almost surely}$$

A proof may be found in [**CT91**]. The SMB can be useful in various situations, for example in analyzing optimal, lossless compression ([**CT91**]).

CHAPTER 9

Risk Minimization and Stochastic Processes

The theorem 5.1 states the correctness of the basic OOM learning algorithm in the limit. That is the algorithm will estimate a "perfect" model almost surely when given an infinite amount of training data. In this regard the theorem is an asymptotic result which presumes a situation of infinite training data supply which is of course not realistic. As it turns out, a learning situation where training data is practically limited to a finite sample is fundamentally different.

Machine learning under the constraint of finite training data is the topic of statistical learning theory ([**Vap98**], [**Vap99**]). Here, desired results are non-asymptotic, quantitative statements about error bounds. In particular, the situation is analyzed from a perspective of risk minimization where "risk" subsumes the expected loss or discrepancy between our model estimate and the hidden target that produced the samples we use for model estimation. Due to space constraints, I can not give an introduction to statistical learning theory here. The reader may find a short introduction in [**Vap99**]. Nevertheless I included this chapter to record some preliminary results and insights that might be of value in future investigations.

The outline of the chapter is as follows. First I argue that a natural choice for the risk functional in the context of estimating distributions of stochastic processes is the relative entropy rate between the estimated model process and the target process. Second, it is shown that under the assumption that the target process is stationary and ergodic, minimizing the risk functional can be done by maximizing the training sample likelihood. Third, under the restriction of finite sample data, to minimize the risk functional it is no longer sufficient to maximize the sample likelihood. Instead, the risk functional is bounded by the sum of the empirical risk and the structural risk which is determined by the model's complexity class. Some comments are given about possible ways of computing the complexity of candidate models in the context of OOM estimation.

9.1. The Risk Functional for Stochastic Processes

In statistical learning theory one assumes that an unknown system has generated the data we observe. The system is supposed to generate samples s independently and subject to some distribution P. In the case the unknown system is a stationary stochastic process, the observed data s might be a finite, contigous part of an realization of the unknown process. The sample s is then used to estimate a model. The goal is to choose a model P_α from a class of candidate models $\alpha \in \Lambda$ such that the *expected loss* or discrepancy between P_α and P becomes minimal. The expectation in computing the loss is taken over the distribution P and a natural ansatz might be the following

$$(9.1)\quad R(\alpha) := \lim_{n\to\infty} \sup \int_{w\in\Sigma^n} L(P(w), P_\alpha(w))\, dP(w) = \lim_{n\to\infty} \sup \mathsf{E}_P[L(P(w), P_\alpha(w))]$$

The expected loss is also called *risk functional*. We still need to choose a loss function L. The loss function should measure the discrepancy between P and P_α on a concrete realization w. Assume we choose

$$L(P(w), P_\alpha(w)) := \frac{1}{|w|} \log \frac{P(w)}{P_\alpha(w)} \tag{9.2}$$

Now, one may observe that the risk functional R with above loss function is nothing else than the relative entropy rate between P and P_α

$$R(\alpha) = \bar{h}_{P||P_\alpha} \tag{9.3}$$

Consequently, minimizing the risk functional over all candidate models $\alpha \in \Lambda$ will select the model with the smallest relative entropy rate in relation to the target process. The following computation (this thesis) for the relative entropy rate between two stationary stochastic processes X_t and Y_t distributed according to P and P_α shows that we can even choose a simpler loss function without altering the selected model.

$$\begin{aligned}
&\bar{h}_{P||P_\alpha}((X_t),(Y_t)) = \lim_{n\to\infty} \sup \frac{1}{n} H_{P||P_\alpha}((X_1,\ldots,X_n) \,||\, (Y_1,\ldots,Y_n)) \\
&= \lim_{n\to\infty} \sup \frac{1}{n} D(P(X_1,\ldots,X_n) \,||\, P_\alpha(Y_1,\ldots,Y_n)) \\
&= \lim_{n\to\infty} \sup \frac{1}{n} \mathsf{E}_P \log \frac{P(X_1,\ldots,X_n)}{P_\alpha(Y_1,\ldots,Y_n)} \\
&\overset{(1)}{=} \lim_{n\to\infty} \sup \frac{1}{n} \mathsf{E}_P \left(\log \frac{1}{P_\alpha(Y_1,\ldots,Y_n)} - \log \frac{1}{P(X_1,\ldots,X_n)} \right) \\
&\overset{(2)}{=} \lim_{n\to\infty} \sup \frac{1}{n} \left(\mathsf{E}_P \log \frac{1}{P_\alpha(Y_1,\ldots,Y_n)} - \mathsf{E}_P \log \frac{1}{P(X_1,\ldots,X_n)} \right) \\
&\overset{(3)}{=} \lim_{n\to\infty} \frac{1}{n} \mathsf{E}_P \log \frac{1}{P_\alpha(Y_1,\ldots,Y_n)} - \lim_{n\to\infty} \frac{1}{n} \mathsf{E}_P \log \frac{1}{P(X_1,\ldots,X_n)} \\
&\overset{(4)}{=} \lim_{n\to\infty} \frac{1}{n} \mathsf{E}_P \log \frac{1}{P_\alpha(Y_1,\ldots,Y_n)} \; - \; h((X_t))
\end{aligned}$$

Up to (1): Definition of relative entropy rate and logarithm. (2): The arguments to the logarithms are positive, thus the logs are positive too. Since (X_t) and (Y_t) are stationary, their entropy rates exist (are finite) and hence we may split up the expectation. (3): Again, since (X_t) and (Y_t) are stationary, the sequence of expected values are non-negative, monotonically decreasing and hence both convergent on their own. This allows us to split the limit term and omit the supremum. (4): By definition of joint entropy and entropy rate.

Since $h((X_t))$ is a constant independent of model selection $\alpha \in \Lambda$, we may omit $P(w)$ in the loss function 9.2 obtaining a simpler loss function

$$L(P(w), P_\alpha(w)) := -\frac{1}{|w|} \log P_\alpha(w) \tag{9.4}$$

without altering the model selection in doing so. The obtained loss function corresponds to equation 5 in [**Vap99**] which is the favored loss function for solving the problem of density estimation.

9.2. Minimizing Empirical Risk and Sample Likelihood

The last section derived the risk functional for the problem of estimating the probability distribution of a stochastic process. We saw that a model can be found by minimizing the risk functional with the loss function 9.4. This involves the computation of expectations $\mathsf{E}_P \log \frac{1}{P_\alpha(Y_1,\ldots,Y_n)}$, which is unfortunate since we obviously can not compute an expectation E_P without knowing P.

The problem can be obviated in case P is the distribution of an *ergodic* process. Then we are allowed to omit the expectation

$$\lim_{n\to\infty} \frac{1}{n} \mathsf{E}_P \log \frac{1}{P_\alpha(Y_1,\ldots,Y_n)} \;=\; \lim_{n\to\infty} \frac{1}{n} \log \frac{1}{P_\alpha(Y_1 = w_1,\ldots,Y_n = w_n)} \tag{9.5}$$

almost surely for realizations w from P. Hence, to minimize the relative entropy rate $\bar{h}_{P||P_\alpha}$ or the risk functional $R(\alpha)$ we can likewise maximize the likelihood $P_\alpha(Y_1 = w_1,\ldots,Y_n = w_n,\ldots)$ of our model process (Y_t) generating the observed realization w.

In statistical learning theory, the expected loss or risk functional $R(\alpha)$ is replaced by the *empirical risk functional* $R_{\text{emp}}(\alpha)$ since we do not know P and we usually only have a finite sample $s \in \Sigma^n$ not an infinite realization w. The empirical risk has the following form in our case

$$R_{\text{emp}}(\alpha) \;:=\; -\frac{1}{n} \log P_\alpha(s) \quad s \in \Sigma^n \tag{9.6}$$

The minimization of the empirical risk is called the principle of empirical risk minimization (ERM). Also, the reader may notice that minimizing $R_{\text{emp}}(\alpha)$ for a given finite sample s is nothing else than the maximum likelihood method in the problem of density estimation.

9.3. Structural Risk and Model Complexity

In the last section equation 9.5 established that minimizing the risk functional when estimating stochastic processes amounts to maximizing the likelihood of a process realization. When facing finite training data we likewise may choose to minimize the empirical risk given in equation 9.6. However, in this case we can not longer be sure about the resulting true risk, since we do not have nonasymptotic bounds on the rate of uniform convergence of the empirical risk functional to the risk functional ([**Vap99**]). Put bluntly, a maximum likelihood fit of a finite sample may result in an overfitted model with bad generalization ability. We are still missing an important point, and this is model complexity. Optimal model selection from finite sample data must simultaneously minimize empirical risk and restrict model complexity. In statistical learning theory this is called the principle of *structural risk minimization* (SRM).

The true risk of model α is seen to be (nonasymptotically) bounded by the sum of the empirical risk for the model α and the structural risk introduced by the complexity H of the subclass Λ_k the model α is taken from:

$$R(\alpha) \;\leq\; R_{\text{emp}}(\alpha) \;+\; H(\Lambda_k) \quad \Lambda_k \subset \Lambda,\; \alpha \in \Lambda_k \tag{9.7}$$

The right side of above equation is the guaranteed risk. Given finite data, learning then must be done by choosing the "right" model complexity class Λ_k , thereby controlling structural risk, and only within the chosen class minimize empirical risk by maximizing the likelihood of the sample data.

In statistical learning theory, when the model class Λ is the set of indicator functions for example, the complexity of subsets of Λ is measured using the notion of VC-dimension ([**Vap99**]). We do not have the analog for stochastic processes or even OOMs. What would be needed is a measure of complexity for subsets of OOMs taken from the set of all OOMs of finite dimension over a fixed alphabet. I can only give first clues here. The following might be relevant in definining complexity for OOMs

(1) the dimension m of the (minimal dimensional) OOM
(2) the complexity of indicative and characteristic events

The dimension of a minimal dimensional OOM is obviously a measure of complexity. An OOM of higher dimension can model more complex data and has more free parameters. The complexity of indicative and characteristic events used while estimating an OOM from training data also might be of relevance, though the events are no longer needed and "effective" after the OOM has been estimated. This is further discussed in section 14.3.

CHAPTER 10

Some Open Questions

The application of ideas from information theory to OOMs already gave valuable insights. However, a number of questions were encountered that could not be treated due to space and time constraints. I would like to close the present part of this thesis by giving a short list of those open questions which might be of interest to OOM theory in the form of tasks.

(1) Analyze convergence of entropy rate for non-stationary OOMs. As was stated, the convergence of entropy rate is know for general stationary processes. However, the entropy rate of OOMs might exist even for non-stationary, but say ergodic OOMs.

(2) Give a closed formula for the entropy rate of OOMs. Such a formula is known for Markov chains for example and it would be nice to have it for OOMs too.

(3) Show finiteness of relative entropy rate between arbitrary OOMs. As was stated, such a result is currently only available for HMMs. Given the importance of relative entropy rate as a quality measure of models, this would be important to have.

(4) Show that (ergodic) OOMs cannot have long memory.

(5) Give a closed formula of auto mutual information function for OOMs.

(6) Given an OOM of dimension m that defines a probability distribution P. Let $(A_i)_i,\ i \in \{1,\ldots,m\},\ A_i \subset \Sigma^k$ be characteristic events of the OOM. Let the OOM be in state w_t at time t. What is the probability that in the next z steps beginning with $t+1$ the OOM produces a trajectory which passes through the series of characteristic events $A_{j_1},\ldots,A_{j_z}$ where $j_1,\ldots,j_z \in \{1,\ldots,m\}$? In other words, how to compute

$$P((X_{t+1},\ldots,X_{t+k}) \in A_{j_1}, (X_{t+2},\ldots,X_{t+k+1}) \in A_{j_2},\ \ldots,\ (X_{t+z},\ldots,X_{t+k+z}) \in A_{j_z} \mid w_t) \quad = \quad ?$$

(7) Give a procedure for computing typical (finite) samples of an OOM. That is one or more samples of given length which have highest probability among all samples of that length.

(8) Give a procedure for computing a maximum likelihood OOM given a finite sample. That is compute an OOM of given dimension that gives maximum probability to the provided sample among all OOMs of that dimension.

(9) Develop a measure of complexity ("VC-dimension") for OOMs and a theory of statistical learning for estimating OOMs from finite data.

Part 3

Partitionings and Context Graphs

In the first part of this thesis, it was shown how to compute an OOM from a finite sample by applying the basic OOM learning algorithm. The basis for the algorithm was a procedure that counts the number of occurrences of certain rasters, indicative and characteristic events, within the sample.

In the last part it was shown that the quality of the estimated OOM crucially depends on the specific choice of indicative and characteristic events. Unfortunately, the search space of indicative and characteristic events is obviously of combinatorial nature and of exponential size.

Consequently, it is quite clear that any efficient learning method for OOMs that strives for good models could profit from a *single unified data structure* that brings together

(1) an index of the training sample
(2) a representation of indicative and characteristic events
(3) counting statistics for those events

This part will satify those aims. In particular, I introduce a newly developed data structure, the *context graph* capable of providing efficient access to all past and future contexts within a string. Secondly, I show how to represent indicative and characteristic events by *partition colorings* of context graphs and how to derive counting statistics as a by-product.

CHAPTER 11

Strings

This chapter lays out some notation and notions related to strings in general which is subsequently needed. The notation introduced is similar to a widely used notation in the formal language community (e.g. [**HU79**]) unless otherwise noted.

11.1. Basic Notation

Strings are finite sequences of symbols from a finite alphabet. In particular, let *string*

(1) Σ fixed, finite and non-empty alphabet, the elements of which we call symbols *alphabet, symbol*
(2) Σ^k set of all strings over Σ of length k
(3) $\Sigma^* := \bigcup_{k\geq 0} \Sigma^k$ set of all strings over Σ including the empty string ϵ
(4) $\Sigma^+ := \Sigma^* \setminus \epsilon$ set of all non-empty strings over Σ
(5) $\Sigma^{k*} := \bigcup_{i\geq k} \Sigma^i$ set of all strings over Σ having length at least k
(6) $\Sigma^{*k} := \bigcup_{i\leq k} \Sigma^i$ set of all strings over Σ having length at most k

The notations in (5) and (6) are specific to this thesis. In this text we will only be concerned with fixed alphabets, which are independent of the length of any input string in particular. For technical reasons, we will assume a total ordering $\prec$ on Σ, alphabet order, inducing a lexicographical ordering on Σ^* which we also denote with $\prec$, where by convention $\epsilon \prec w,\ \forall w \in \Sigma^+$. *alphabet order*

Occasionally, we will talk about *infinite* sequences of symbols from a finite alphabet. These entities will be referred to just as sequences.

sequence

$$\Sigma^\infty := \{\tau \mid \tau : \mathbb{N} \longrightarrow \Sigma\} \quad \text{set of all (infinite) sequences over } \Sigma$$

We use w, v, u, r, s, t to denote words or strings over Σ and x, y, z to denote single symbols from the alphabet. A reversed string is written s^{-1}. Further, let *reversed string*

(1) $\text{len}(s) := |s| =$ length of string s
(2) for $i, j \in \{1, \ldots, \text{len}(s)\}$
    (a) $s[i:j]$ factor (substring) of s starting from position i inclusive and ending with position j inclusive *factor*
    (b) $s[i:] := s[i : \text{len}(s)]$ suffix of s starting from position i inclusive *suffix*
    (c) $s[:i] := s[1:i]$ prefix of s ending with position i inclusive *prefix*

That is, symbol positions within strings are numbered beginning with 1, designating the first symbol in the string, up to $\text{len}(s)$ designating the last symbol in the string. Further, we agree on $s[i:j] = \epsilon$ whenever $i > j$. We use the notation $t \sqsubseteq s$ when t is a factor of s. A factor, prefix or suffix of a string w is called *proper* if it is different from w itself. *proper factor, suffix or prefix*

The sets of all factors (substrings), prefixes and suffixes of a given string s are defined as

DEFINITION 11.1 (Factorset, Prefixset, Suffixset (from [**IHS$^+$01a, IHS$^+$01b**])).

$$\mathsf{Factor}(s) := \bigcup_{i,j\in\mathbb{N}} \{s[i:j]\}$$
$$\mathsf{Prefix}(s) := \bigcup_{i\in\mathbb{N}} \{s[:i]\}$$
$$\mathsf{Suffix}(s) := \bigcup_{i\in\mathbb{N}} \{s[i:]\}$$

right-branching *left-branching*

Note, that both ϵ and s are elements of all three sets. Trivially, $\mathsf{Prefix}(s) \subseteq \mathsf{Factor}(s)$ and $\mathsf{Suffix}(s) \subseteq \mathsf{Factor}(s)$ where the containment is strict iff $|s| > 1$. A factor $w \in \mathsf{Factor}(s)$ of s is called *right-branching* (*left-branching*) if there exist $x, y \in \Sigma$, $x \neq y$ such that $wx, wy \in \mathsf{Factor}(s)$ ($xw, yw \in \mathsf{Factor}(s)$).

nested prefix or suffix

A prefix $v \in \mathsf{Prefix}(s)$ of s is called *nested*, if there exists a $wv \in \mathsf{Prefix}(s)$ with $w \neq \epsilon$. Thus, a nested prefix of s is itself a proper suffix to a different prefix of s. Similarly, a suffix $u \in \mathsf{Suffix}(s)$ of s is called nested, if there exists a $uw \in \mathsf{Suffix}(s)$ with $w \neq \epsilon$. A nested suffix of s is itself a proper prefix to a different suffix of s.

11.2. Equivalence Relations on Strings

The notions introduced in the following may seem rather artificial first, but later on will allow us to systematically introduce and x-ray suffix trees, a data structure of fundamental importance to this text. We will follow [**IHS$^+$01a, IHS$^+$01b**] here. Given a fixed string $s \in \Sigma^+$, we introduce two binary relations on Σ^*.

DEFINITION 11.2.

$w \equiv_s^L r \quad :\Leftrightarrow$ the set of positions in s at which w and r *begin* are identical

$w \equiv_s^R r \quad :\Leftrightarrow$ the set of positions in s at which w and r *end* are identical

Obviously, the relations are equivalence relations. Given $w \in \Sigma^*$ the equivalence classes containing w with respect to $\equiv_s^L$ and $\equiv_s^R$ will be denoted $[w]_s^L$ and $[w]_s^R$ respectively.

Given $s \in \Sigma^+$, all strings **not** factors of s form one equivalence class under $\equiv_s^L$ and $\equiv_s^R$ each.

EXAMPLE 11.1. Suppose $\Sigma = \{a, b, e, g\}$ and $s = baggage$. Then $[a]_s^L = \{a, ag\}$, $[a]_s^R = \{a\}$, $[ga]_s^L = \{ga, gag, gage\}$ and $[ga]_s^R = \{ga, gga, agga, bagga\}$.

EXAMPLE 11.2. Suppose $\Sigma = \{a, b\}$ and $s = aabbabbbaabbabaabbba$. Then $[aab]_s^L = \{aa, aab, aabb\}$ and $[aab]_s^R = \{aab\}$.

EXAMPLE 11.3. Suppose $\Sigma = \{a, c, o\}$ and $s = cocoa$. Then $[\epsilon]_s^R = \{\epsilon\}$, $[c]_s^R = \{c\}$, $[o]_s^R = \{o, co\}$, $[a]_s^R = \{a, oa, coa, ocoa, cocoa\}$ $[oc]_s^R = \{oc, coc\}$ and $[oco]_s^R = \{oco, coco\}$. Further all $w \in \Sigma^*$ not already member of one of the listed equivalence classes form one class.

By definition of $\equiv_s^{L|R}$ and given $w \in \Sigma^*$, $r_1, r_2 \in [w]_s^{L|R}$ then either r_1 is prefix (suffix) of r_2 or vice versa. Hence, we always find a unique longest member $\overrightarrow{\overset{s}{w}}$ and $\overleftarrow{\overset{s}{w}}$ in every equivalence class $[w]_s^L$ and $[w]_s^R$, which will serve as the *representative* of the respective class. In other words

$$[w]_s^L \subset \mathsf{Prefix}(\overset{s}{\overrightarrow{w}}) \tag{11.1}$$

$$[w]_s^R \subset \mathsf{Suffix}(\overset{s}{\overleftarrow{w}}) \tag{11.2}$$

CHAPTER 12

Partitionings

In this chapter I introduce partitionings, partition functions and a natural equivalence relation on the space of partitionings and partition functions. These notions have been developed in the present thesis with the aim of being able to compare all possible choices of characteristic or indicative events within a unified space of partitionings.

The chapter will follow this outline: First, partitionings of Σ^k are defined using two equivalent notions. Second, the domains of those partitionings are extended in two steps to the full string space Σ^* which allows us to directly compare them in a unified space. Third, we study the space of all such extended partitionings with respect to some given and fixed $s \in \Sigma^*$ and derive a natural equivalence relation. This equivalence relation will later be rediscovered within the context of partition colored suffix trees.

12.1. Partitionings and Partition Functions

The idea of decomposing a set of strings Σ^k into a familiy of disjoint sets can be formalized using a set- or a function-oriented notion. A set-oriented formulation is

m-partitioning

DEFINITION 12.1. A *m-partitioning* of Σ^k where $m \in \mathbb{N},\ m \geq 2$ is a family $\{A_i\}_{i\in\{1,..,m\}}$, $A_i \subset \Sigma^k$ such that

(1) $\bigcup_{i\in\{1,..,m\}} A_i = \Sigma^k$
(2) $A_i \cap A_j = \emptyset \quad \forall i \neq j \quad i,j \in \{1,..,m\}$
(3) $A_i \neq \emptyset \quad \forall i \in \{1,..,m\}$

Requirement (3) ensures that the m-partitioning does not degenerate. A function-oriented formulation of the same idea is

m-partition function

DEFINITION 12.2. A *m-partition function* π of Σ^k is a membership function

$$\pi : \Sigma^k \longrightarrow \{1,\ldots,m\}$$

such that π is surjective.

Obviously, m-partitionings of Σ^k and m-partition functions of Σ^k are formally equivalent and we will use the one over the other where it is convenient to do so.

Also, we use shorthands of the form $\pi = (A_i)_{i\in\{1,\ldots,m\}}$ to specify m-partition functions where A_i are sets as with m-partitionings.

12.2. Closures of Partition Functions

We proceed with step 2 of the chapter outline, extending the domain of partitionings and partition functions.

Given some m-partition function π of Σ^k one can extend the domain of the function to Σ^{k*} using some *outer closure* operator oCl:

DEFINITION 12.3. Given a m-partition function π of Σ^k, the *outer closure* of π is given by *outer closure*

$$\mathsf{oCl}(\pi) : \Sigma^{k*} \longrightarrow \{1, \ldots, m\}$$

where

$$\mathsf{oCl}(\pi)(s) \;:=\; \pi(s[:k])$$

The outer closure $\mathsf{oCl}(\cdot)$ plainly assigns values exactly as the original partition function does on k-prefixes of the given string.

Now, it is natural to ask if, for a given m-partition function π of Σ^k, there is a $l < k$ and a m-partition function $\tilde{\pi}$ over Σ^l such that

$$\mathsf{oCl}(\tilde{\pi})(s) = \pi(s) \quad \forall s \in \Sigma^k \tag{12.1}$$

If this is the case, we call π *degenerate* as $\tilde{\pi}$ is equivalent to π but is specified on the smaller set Σ^l. *degenerate partitioning*

EXAMPLE 12.1. Suppose $\Sigma = \{a, b\}$ and you are given a 2-partition function π of Σ^2 defined as $\pi(aa) = 1,\ \pi(ab) = 1,\ \pi(ba) = 2,\ \pi(bb) = 2$. Obviously, π is degenerate as there is a 1-partition function $\tilde{\pi}$ of Σ^1 with $\tilde{\pi}(a) = 1,\ \tilde{\pi}(b) = 2$ that has a outer closure $\mathsf{oCl}(\tilde{\pi})$ and takes identical values on Σ^k as π itself.

This last observation suggests the

DEFINITION 12.4. Given a m-partition function π of Σ^k, the *depth* of π is given by *depth of partition function*

$$\mathsf{depth}(\pi) := \max_{s \in \Sigma^{*(k-1)}} \{|s| + 1 \mid \exists w, w' \in \Sigma^{k-|s|} : \pi(sw) \neq \pi(sw')\}$$

Obviously, if for a given m-partition function π of Σ^k the $\mathsf{depth}(\pi) < k$, then π is degenerate. Intuitively, it is wasted effort to specify a partition function for Σ^k if we could do the same as with outer closure of some partition function of $\Sigma^{k'}$ where $k' < k$.

Also, the depth of a partition function can be thought of as a possible measure of complexity of the thus defined partitioning.

EXAMPLE 12.2. Suppose $\Sigma = \{a, b\}$ and $\pi = (A_1, A_2)$ is a 2-partition function of Σ^3 defined by $A_1 = \{aaa, aab\}$ and $A_2 = \{a, b\}^3 \setminus A_1$. Then $\mathsf{depth}(\pi) = 2$.

Our immediate goal now is to extend the domain of m-partition functions π of Σ^k to the whole Σ^*. The outer closure $\mathsf{oCl}(\pi)$ takes us only half the way on this road. What we need is

DEFINITION 12.5. Given a m-partition function π of Σ^k, the *inner closure* of π is given by *inner closure*

$$\mathsf{iCl}(\pi) : \Sigma^{*k} \longrightarrow \{0, 1, \ldots, m\}$$

$$\mathsf{iCl}(\pi)(s) := \begin{cases} \pi(s\cdot) & \text{if} \quad \forall \omega, \omega' \in \Sigma^{k-|s|} \;:\; \pi(s\omega) = \pi(s\omega') \\ 0 & \text{else} \end{cases}$$

What we added is a pseudo partition 0 to the existing set of partitions $\{1, \ldots, m\}$. This pseudo partition 0 corresponds to neutrally colored nodes in a partition colored suffix tree - a concept we will introduce later.

We now have inner and outer closures of m-partition functions and can combine both to form the complete closure

complete closure

DEFINITION 12.6. Given some arbitrary m-partition function π of Σ^k, we define the *closure* of π to be

$$\mathsf{Cl}(\pi) : \Sigma^* \longrightarrow \{0, 1, \ldots, m\}$$

with

$$\mathsf{Cl}(\pi)(s) \;=\; \begin{cases} \mathsf{iCl}(s) & \text{if } |s| \leq k \\ \mathsf{oCl}(s) & \text{else} \end{cases}$$

Because of all this, beginning from here we will just speak of the closure $\mathsf{Cl}(\pi)$ of a partition function π.

12.3. Equivalence of Partitionings

With the help of the closure operator just introduced, we are now ready to directly compare *all* partition functions in one unified space of partition functions

partition space

DEFINITION 12.7. Given an alphabet Σ and $m \in \mathbb{N}$, the m-partition space over Σ is given by

$$\Pi_m(\Sigma) := \bigcup_{k \in \mathbb{N}} \{\mathsf{Cl}(\pi) : \pi \text{ is } m\text{-partition function of } \Sigma^k\}$$

Beginning from now, when Σ is agreed upon, the elements of Π_m will plainly be called m-partition functions or m-partitionings.

The final stage 3 of our initial chapter outline involves comparing partition functions due to some suitable equivalence relation, when given a fixed string $s \in \Sigma^+$.

Given a fixed string $s \in \Sigma^+$ and a fixed dimension $m \in \mathbb{N}$, we introduce a binary relation $=_s$ on the partition space $\Pi_m(\Sigma)$

DEFINITION 12.8. Two partitionings $\pi, \pi' \in \Pi_m(\Sigma)$ are *equivalent* with respect to s

$$\pi \;=_s\; \pi'$$

iff

$$\forall w \sqsubseteq s \;:\; \pi(w) = \pi'(w)$$

This makes sense, since given a fixed string s, two partition functions can only be distinguished up to a point where they take identical values on all substrings of our reference s. If this is the case, we identify both. In other words

LEMMA 12.1. $=_s$ is an equivalence relation.

PROOF. Obviously, $=_s$ is reflexive and symmetric. The relation is also transitive since $\pi_1(w) = \pi_2(w) \;\wedge\; \pi_2(w) = \pi_3(w)$ implies $\pi_1(w) = \pi_3(w)$. □

We complete our notation by

DEFINITION 12.9. Given $s \in \Sigma^*$ and a m-partition function $\pi \in \Pi_m(\Sigma)$. Then the *equivalence class* of π with respect to s is given by

$$[\pi]_s \subset \Pi_m(\Sigma)$$

with

$$\pi' \in [\pi]_s \ :\Leftrightarrow \ \pi' =_s \pi$$

EXAMPLE 12.3. Given $\Sigma = \{a, b\}$ and $s = abb$. Then the 3-partition functions $\pi_1 = (\{aa\}, \{ab, bb\}, \{ba\})$ and $\pi_2 = (\{ba\}, \{ab, bb\}, \{aa\})$ are equivalent with respect to s, that is $\pi_1 =_s \pi_2$. This is the case since $\pi_1(a) = \pi_2(a) = 0$, $\pi_1(b) = \pi_2(b) = 0$, $\pi_1(ab) = \pi_2(ab) = 2$ and $\pi_1(bb) = \pi_2(bb) = 2$.

CHAPTER 13

Suffix Trees

13.1. An Informal Introduction

Exposing the internal structure of individual strings or making a set of strings accessible is important for effectively solving many practical problems. The search for good indicative and characteristic events when given a finite training sample is one application. This section is a gentle preface to the formal treatment of the topic in later sections.

trie

Historically, one of the earliest notions in the area was that of a trie. A *trie* (from re**trie**val), is a multi-way (k-adic) tree structure useful for storing strings over an alphabet Σ of size k. The concept is usually attributed to Fredkin in 1960 [**Fre60**].

Edges are labeled with symbols from the alphabet. The crucial point is, that all edges leaving some node must start with a different symbol. This facilitates the idea that all stored strings sharing some prefix should hang off a common node. Strings are then inserted by creating new nodes and edges as necessary such that a string can be recovered by traveling from the root to some node and writing down the concatenation of edge labels. All strings stored in a trie can be recovered by a depth-first scan of the tree.

Patricia tree

A *Patricia tree* is like a trie, but with non-branching internal nodes collapsed or compacted, such that every node other than the root or a leaf has at least two children. Patricia trees were introduced as an index structure for searching in marked-up text by Morrison in the late 1960s [**Mor68**] as an improvement over tries. Treatments of Patricia trees can be found in many data structures textbooks.

atomic suffix tree
compact suffix tree

An *atomic suffix tree* of a string $s \in \Sigma^n$ is a trie built over all suffixes of s, whereas a *compact suffix tree* of a string $s \in \Sigma^n$ is a Patricia tree built over all suffixes of s. Thus, atomic and compact suffix trees are special cases of tries and Patricia trees where the set of strings stored is the suffix set of a single string. However, the crucial point is that the suffixes of a given string are not independent. This, as it turns out, has many rich and powerful consequences.

Trivial examples of atomic and compact suffix trees are given in figure 13.1. The figure also sketches interesting connections between two other closely related data structures, discussed further below.

DAWG

A *Directed Acyclic Word Graph* (DAWG) of a string s specifies the smallest finite state automaton (FSA) that recognizes all suffixes of s. It can be built from s in time $\mathcal{O}(\text{len}(s))$ and stored in space $\mathcal{O}(\text{len}(s))$. The storage requirements of the atomic suffix tree of s is of $\mathcal{O}(\text{len}(s)^2)$ as is the time complexity to build it.

compact DAWG

The *compact DAWG* of a string s is simply the edge compressed or compacted DAWG of s.

We now discuss some further aspects of suffix trees. Since every substring of a string s is a prefix of some suffix of s, a suffix tree of s contains all substrings of s.

If $\text{len}(s) = n$, then there are $n \cdot (n+1)/2$ substrings in s. Hence, an *atomic* has suffix tree at most $\mathcal{O}(n^2)$ nodes. Of course, the substrings of s do not have to be all different.

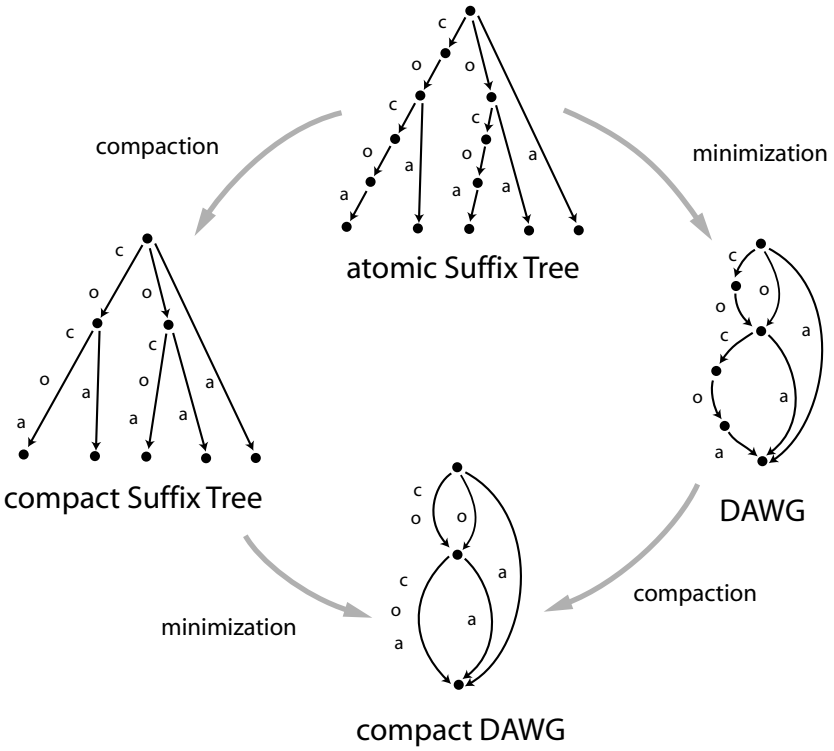

FIGURE 13.1. Relationship among atomic suffix tree, compact suffix tree, DAWG and compact DAWG for string 'cocoa'. (adapted from [**IHS+01b**])

Appending just one symbol at the end of s adds $n+1$ substrings, but again these are not independent.

In contrast, *compact* suffix trees can be built in both space and time complexity $\mathcal{O}(n)$. The linear space complexity can be achieved by storing edges in some compressed form, e.g. only storing indexes into the input string instead of storing the actual substring associated with the edge. The linear time complexity is really surprising and indeed the first algorithm to meet this bound was given in the 1970s by Weiner [**Wei73**]. Later improvements are due to McCreight [**McC76**]. Recently, another linear run-time algorithm was invented by Ukkonen [**Ukk95**] with the additional property of reading the input strictly left-to-right and producing suffic trees already in every intermediate step, the so-called *on-line* property, which is important for some applications. It must be noted that the linear time algorithms for compact suffix tree construction are non-trivial and not simple to grasp. This might be one reason that suffix trees today are virtually never treated in standard data structures textbooks, which is quite astonishing given their power.

I found the following references especially valuable introductions into the topic of suffix trees [**Gus97**], [**GK97**]. Today, the research in suffix tree algorithms includes improving the constant in linear time complexity and, also important, improving the constant in storage needed ([**GKS99**]). Other directions of research are persistent and external storage suffix trees ([**FFM**]) and large alphabets ([**Far96**]). A topic which is very intersting but cannot be treated here for reasons of limited resources is asymptotic properties of suffix trees under probabilistic assumptions for the distribution of input strings ([**Szp93b**], [**Szp93a**]). Another fascinating direction are *affix trees* which are suffix tree extensions such that the resulting structure is self-dual ([**Sto95**], [**Maa00**]).

13.2. Applications of Suffix Trees

Its probably fair to say that suffix trees are among the most versatile data structures currently known. Applications range from exact, approximate and regular expression string matching, general string searching and processing to biosequence analysis. An extensive treatment of the just mentioned applications can be found in [**Gus97**], [**Apo85**].

Other, more unusual applications include data compression ([**Lar98**], [**BK00**]) and accelerating string kernels for Support Vector Machine (SVM) based text classification [**LEN02**].

Here are four problems from the string processing domain effectively solvable using suffix trees. Of course there are many more.

(1) *String Search* Searching for a pattern p in a text s can be solved in $\mathcal{O}(\text{len}(s))$ for preprocessing the string s *once*, which means building the suffix tree, and then for every search of a pattern p a time of $\mathcal{O}(\text{len}(p))$. The search is performed by traversing the suffix tree beginning from the root, along edges matching the pattern string searched for. If all symbols of the patterns have been eaten up without a mismatch, then and only then the search pattern occurs in the text.

(2) *Longest Repeated Substring* A longest repeated substring is a substring occuring at least twice, having a length that is maximal among those reoccurring substrings. An instance can be found by looking for a deepest forking node in the suffix tree, where depth is measured by the number of symbols traversed from the root along the way to the forking node. This can be done in $\mathcal{O}(n)$.

(3) *Longest Common Substring* of two strings s_1 and s_2 can be found by building the suffix tree for $s_1\$s_2\#$ where \$ and # must not occur in s_1 and s_2. Then, look for the deepest forking node which has both a continuation where only # appears in (a substring of s_2) and a continuation where \$ appears in (a prefix of a substring of s_1). Again, this can be done in $\mathcal{O}(n)$.

(4) *Longest Palindrome* A palindrome of s is a substring ω such that $\omega = \omega^{-1}$. A longest palindrome can be found by building the suffix tree for $s\$s^{-1}\#$ and looking for a deepest forking node that satifies the criteria like in Longest Common Substring.

13.3. Σ^+-trees

This and the next section reintroduces suffix trees and related notions, this time formally. The results and notation follow [**GK97**] except proposition 13.2 which was derived in this thesis.

DEFINITION 13.1. A Σ^+-tree T is a tree with edges labeled from Σ^+ such that no two edges leaving some node k of T have labels beginning with the same symbol $a \in \Sigma$. *Σ^+-tree*

Because of this special uniqueness property of outgoing edge labels, we may speak of the *path* to a node k of T, defined as the concatenation of all edge labels encountered when traveling from the root to k. Since paths of nodes are unique themself, we can identify the node k with $\bar{w}$ iff $\mathsf{path}(k) = w$. *path*

DEFINITION 13.2. A Σ^+-Tree is T called *atomic* if all edges in T are labeled with exactly one symbol each. A Σ^+-Tree T is called *compact* if all nodes in T are either leaves or have at least two children. In other words, all internal nodes are branching. *atomic and compact Σ^+-trees*

The nodes $\bar{w}$ where $\exists k \in T : w = \mathsf{path}(k)v$ are called *explicit* nodes when $v = \epsilon$ and are called *implicit* nodes when $v \neq \epsilon$. In the latter case, the "nodes" reside within edges. Obviously, in an atomic suffix tree all nodes are explicit. *explicit nodes* *implicit nodes*

It seems natural to ask what symbol sequences we can encounter on any journey beginning at the root of a Σ^+-Tree, which leads to the following

DEFINITION 13.3. Given a Σ^+-Tree T, then $\mathsf{Words}(T)$ is defined Words

$$w \in \mathsf{Words}(T) \quad :\Longleftrightarrow \quad \exists k \in T : \; w = \mathsf{path}(k)v$$
where v is a prefix of an outgoing edge of k

When $w \in \mathsf{Words}(T)$, then the *reference node* of w, denoted $\mathsf{ref}(w)$, is the unique node in T such that $\mathsf{path}(\mathsf{ref}(w))$ is the longest prefix of w among all nodes of T. It will then hold that $w = \mathsf{path}(\mathsf{ref}(w))v$ where v is a prefix of an outgoing edge of $\mathsf{ref}(w)$. *reference node*

Further we use the notations $\mathsf{Leaves}(T)$ to denote the set of all leaves in a tree T, $\mathsf{Leaves}(k)$ to denote the set of all leaves in T under a node k, $\mathsf{Descs}(k)$ to denote the set of all descendants under node k including k itself, $\mathsf{Children}(k)$ to denote all direct children of k (if any) and $\mathsf{Parent}(k)$ the parent node of node k with $\mathsf{Parent}(k) = k$ if k is the root. Leaves Descs Children Parent

13.4. Atomic and Compact Suffix Trees

Now, a *suffix tree* $T(s)$ of some string s is simply a Σ^+-Tree such that *suffix tree*

$$\mathsf{Words}(T(s)) = \mathsf{Factor}(s). \tag{13.1}$$

An atomic suffix tree of s will be denoted $\mathsf{ast}(s)$. A compact suffix tree of s will be denoted $\mathsf{cst}(s)$ instead. Note, that a suffix tree of s is not uniquely determined in general, but the atomic and compact forms are.

A suffix tree of s^{-1} is a *reverse prefix tree* of s ([**GK97**]). This is easy to see if we observe that every suffix in s^{-1} is equal to a prefix of s when the prefix is reversed. The atomic and compact forms of reverse prefix trees of s will be denoted $\mathsf{apt}(s) = \mathsf{ast}(s^{-1})$ *reverse prefix tree*

and $\mathsf{cpt}(s) = \mathsf{cst}(s^{-1})$ respectively.

If s has a suffix s_1 which is itself prefix to another suffix s_2 of s, then s_1 cannot be represented by a leaf in any suffix tree of s. However, sometimes it is desireable to have every suffix of s represented by a leaf. Thinking about it, one observes that this can be enforced by appending a *sentinel* symbol \$, a symbol which does not occur anywhere else in s. Then, every suffix of $s\$$ and only the suffixes of s are represented as leaves.

sentinel

Examples of the compact suffix tree and the reverse prefix tree for the same string $s = aabbabbbaabbabaabbba \in \{a, b\}^{20}$ extended by a sentinel \$ are given in figures 13.5 and 13.5. The graphics were automatically generated as a byproduct by the suffix tree analysis tool sftree which I developed during this thesis. The tool is described in detail in appendix 20.

The following two lemmata are merely reformulations of the discussion in [**GK97**] to help in understanding suffix trees.

LEMMA 13.1. Let w be a factor of s and T a suffix tree of s. Then all factors of s having w as a prefix can be found in the subtree of T hanging off $\mathsf{ref}(w)$ or the parent or ancestors of $\mathsf{ref}(w)$.

PROOF. Since edges leaving a node are uniquely distinguished by the first symbol on their edge label, all positions in T found by traversing the tree via label matching are unique. Further, every so found position also represents a string which is a substring of s. Finally, given a position in T that represent w, all positions downwards the tree represent longer strings wv. □

LEMMA 13.2. Let w be a factor of s and T a suffix tree of s. Then, if no suffix of s is nested, the number of occurrences of w in s is given by the leaf count in the substree of T hanging off $\mathsf{ref}(w)$.

PROOF. Since no suffix of s is nested, a suffix tree for s will have exactly $|s|$ leaves, one for each suffix of s. Further, observe that every substring of s is a prefix of some suffix of s. Hence, we just have to count all the suffixes of s that have the prefix w. But those suffixes with prefix w are all in the subtree specified as the previous lemma has shown and are represented by leaves therein. □

The precondition that no suffix of s is nested can be reached simply by appending a sentinel $\$ \notin \Sigma$ to s. In other words, the string $s\$$ has no nested suffixes as \$ by definition does not occur anywhere in s.

13.5. Suffix Links

For the construction of suffix trees and various applications a special kind of auxiliary edges connecting nodes in a suffix tree is important. These edges are in addition to the normal tree edges.

suffix link

DEFINITION 13.4. (from [**GK97**]) Given a Σ^+-tree T and a node $x\bar{w}$ in T. Let v be the longest suffix of w such that $\bar{v}$ is a node in T and let $xw = uv$. Then an edge $x\bar{w} \to \bar{v}$ is called a *suffix link* in T. A suffix link $x\bar{w} \to \bar{w}$ (that is $u = x$ and $v = w$) is called *atomic*.

Thus, the suffix link of $x\bar{w}$ is pointing to a node with a path label v that is derived of xw by separating the shortest possible prefix u.

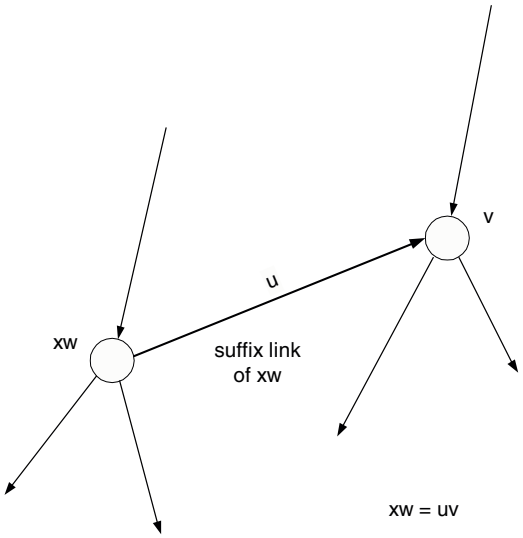

FIGURE 13.2. A suffix link.

Also, the node $\bar{v}$ is well-defined since ϵ is a suffix of w and $\bar{\epsilon}$ is a node in T (the root). An example of a suffix link is given in figure 13.2. Note, that suffix links are sometimes defined as unlabeled edges. Alternatively, suffix links $x\bar{w} \to \bar{v}$, where $xw = uv$ may be labeled by the prefix u they separate.

The following propositions state some interesting facts about suffix links.

PROPOSITION 13.1. (from [**GK97**])

(1) In the atomic suffix tree of s all suffix links are atomic.
(2) In the compact suffix tree of $s\$$ where $\$ \notin \Sigma$, all suffix links are atomic.

PROOF. (1): Follows from the definition of atomic suffix trees, since in $\mathsf{ast}(s)$ all nodes are explicit. (2): $x\bar{w}$ is either a branching node or a leaf. Thus, xw is right-branching or a non-nested suffix of $s\$$. But then the same holds true for w and so $\bar{w}$ must also be an explicit node in $\mathsf{cst}(s\$)$. □

PROPOSITION 13.2. (this thesis) In the compact suffix tree of $s\$$ where $\$ \notin \Sigma$, all suffix links of leaves again point to leaves.

PROOF. Let $k \in \mathsf{Leaves}(\mathsf{cst}(s\$))$. Then $\mathsf{path}(k) = xw\$$ for some $w \in \Sigma^*,\ x \in \Sigma$, the suffix link of k is atomic (proposition 13.1(2)) and points to a node l with path label $w\$$. Hence l is a leaf. □

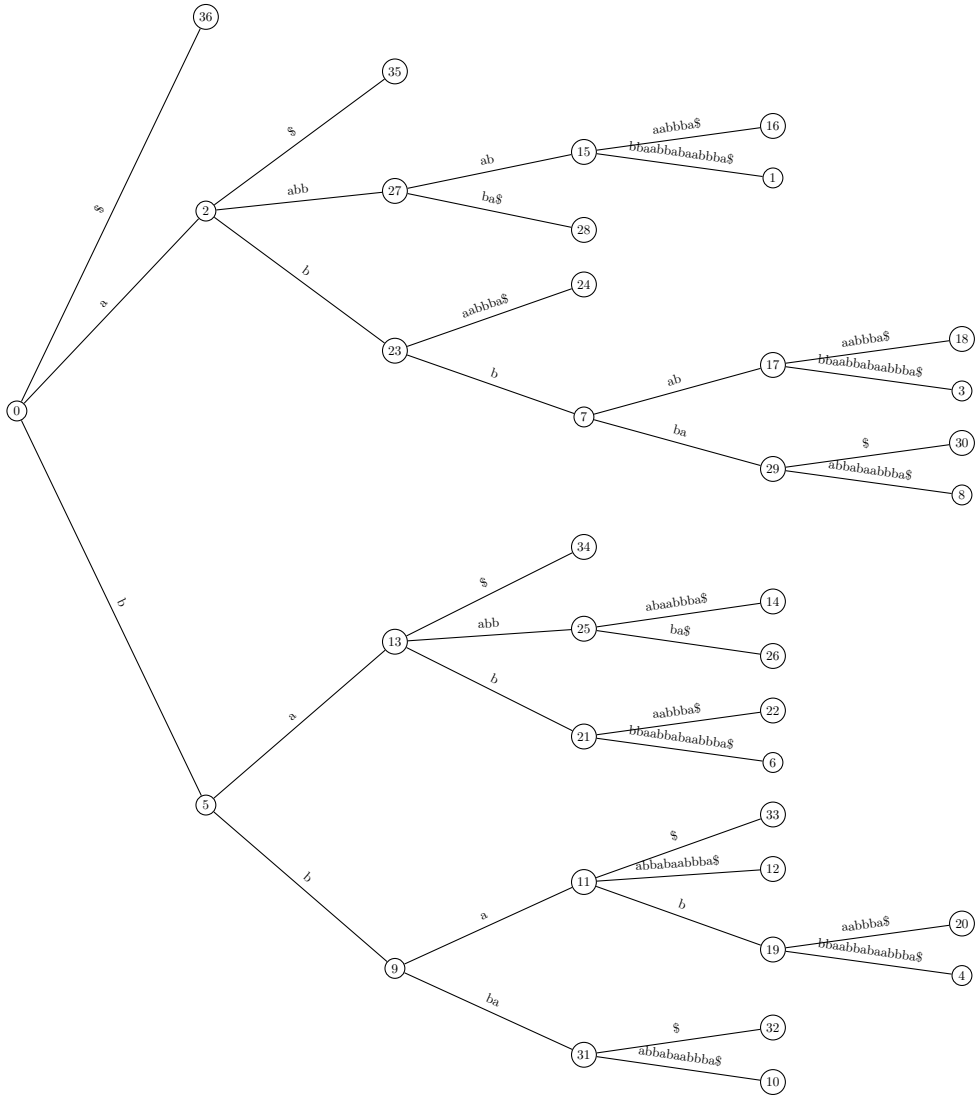

FIGURE 13.3. Compact suffix tree for $s = aabbabbbaabbabaabbba\$$.

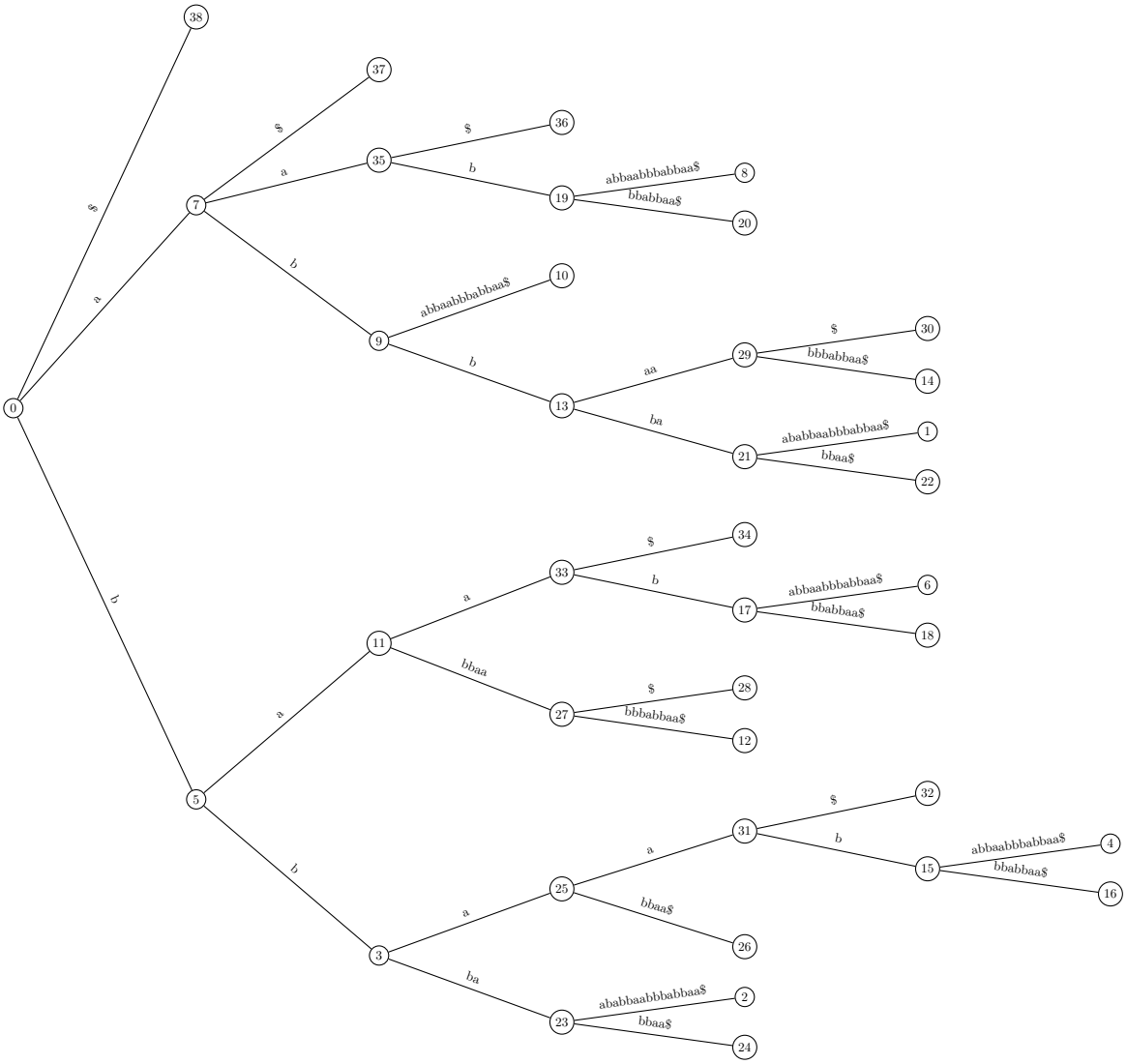

FIGURE 13.4. Compact reverse prefix tree for $s = \$aabbabbbaabbabaabbba$.

CHAPTER 14

Partition Colorings of Suffix Trees

In section 12.3 we saw that with respect to a fixed string $s \in \Sigma^*$ the set of all m-partition functions $\Pi_m(\Sigma)$ resolves into equivalence classes. Recall, this was the case as s simply does not contain every string $t \in \Sigma^*$ and thus partition functions only differing in the classes they assign to t will be indifferent with respect to s.

How can we effectively represent the s-distinguishable m-partition functions? As it turns out, there is a one-to-one relation between certain colorings of compact suffix trees and equivalence classes of partition functions. All results and notions in this chapter were derived in this thesis.

14.1. Partition Colorings

m-partition coloring

DEFINITION 14.1. Given a string $s \in \Sigma^*$, a sentinel $\$ \notin \Sigma$ and $m \in \mathbb{N}$. Then a *m-partition coloring* of $T = \mathsf{cst}(s\$)$ is a mapping

$$\pi \ : \ T \ \rightarrow \ \{0, \ldots, m\}$$

such that

(1) π is surjective
(2) $\pi(l) > 0$ for all $l \in \mathsf{Leaves}(T)$.
(3) $\pi(k) > 0 \ \Rightarrow \ \pi(k') = \pi(k) \quad \forall k' \in \mathsf{Children}_T(k)$

Similar to definition 12.5, if $\pi(k) = 0$ we say the node k is neutrally colored. The conditions assure that (1) the coloring is non-degenerate, (2) all leaves are colored by a non-neutral color and (3) once a node is colored non-neutrally, then all its children and thereby all its descendents have the same color. Examples of partition colorings of a suffix tree are given in the figures 14.1 and 14.2. Note that in these figures the edge labels have been written not along the edges between nodes but within the nodes. That is the label of an edge to a parent node is written in the node itself. Further, the index pairs give start and end indices into the input string. For example "$abb(1, 3)$" means that during suffix tree construction the edge label abb has been constructed from $s[2, 4]$. The indices must be incremented by one, since for historical reasons in the figures the indices were taken to start at 0.

It is important to note that the top-down propagation of node colorings by condition (3) in above definition 14.1 has a simple consequence

PROPOSITION 14.1. A m-partition coloring π of a suffix tree $T = \mathsf{cst}(s\$)$ is already fully specified by giving

$$\pi(l) \quad \forall l \in \mathsf{Leaves}(T)$$

that is the coloring of all leaves of T.

PROOF. Given that π is already defined on all leaves of T we have to define π on all inner nodes. This can be done recursively by: if $\pi(k) = \pi(k')$ for all $k, k' \in \mathsf{Children}(\mathsf{Parent}(k))$ then define $\pi(\mathsf{Parent}(k)) := \pi(k)$. Otherwise define $\pi(\mathsf{Parent}(k)) := 0$. □

The proposition shows, that if we are given a m-partition colored suffix tree it is sufficient for us to know the colorings of the leaves to know the complete m-partition coloring. However, not every coloring of leaves gives rise to a valid partition coloring, as the following proposition clearifies.

PROPOSITION 14.2. A coloring π of the leaves of a suffix tree $T = \mathsf{cst}(s\$)$ induces a m-partition coloring π of T iff

(1) $\pi(l) > 0 \quad \forall l \in \mathsf{Leaves}(T)$
(2) $\forall r \in \{1, \ldots, m\} \ : \ \exists l \in \mathsf{Leaves}(T) : \pi(k) = r$

PROOF. This is easy to see from the previous proposition and the fact that π is surjective in $\{1, \ldots, m\}$ iff π is surjective in $\{1, \ldots, m\}$ on the leaves of T. □

In other words, a coloring of the leaves of a suffix tree gives rise to a uniquely determined partition coloring iff all leaves are colored non-neutrally and every color $\{1, \ldots, m\}$ occurs.

14.2. Equivalence of Partition Colorings and Partition Functions

Now, given $s \in \Sigma^*$ and a m-partition function $\pi \in \Pi_m(\Sigma \cup \{\$\})$ such that

$$\pi(s[i:]\$) > 0 \ \forall i \tag{14.1}$$

$$\forall r \in \{1, \ldots, m\} \ \exists i \ : \ \pi(s[i:]\$) = r \tag{14.2}$$

The conditions assure that π maps all suffixes of $s\$$ surjectively to $\{1, \ldots, m\}$. Then we can define a coloring $\tilde{\pi}$ of $\mathsf{cst}(s\$)$.

$$\tilde{\pi}(k) = \pi(\mathsf{path}(k)) \quad \text{for all} \quad k \in \mathsf{Nodes}(\mathsf{cst}(s\$)) \tag{14.3}$$

LEMMA 14.1. $\tilde{\pi}$ is a m-partition coloring of $\mathsf{cst}(s\$)$

PROOF. By construction π maps all suffixes of $s\$$ surjectively into $\{1, \ldots, m\}$. Hence, as the compact suffix tree of $s\$$ has exactly one leaf per suffix of $s\$$, $\tilde{\pi}(l) > 0$ for all leaves l. Further note, that by definition of iCl, $\pi(w) > 0$ implies $\pi(wr) = \pi(wr') \ \forall r, r'$ which directly translates to $\tilde{\pi}$, that is $\tilde{\pi}(\bar{w}r) = \tilde{\pi}(\bar{w}r') \ \forall r, r'$. □

Further observe that

LEMMA 14.2. If $\pi_1, \pi_2 \in [\pi]_{s\$}$ are two m-partition functions equivalent with respect to $s\$$ (and fulfilling the conditions 14.1 and 14.2), then the m-partition colorings of $\mathsf{cst}(s\$)$ induced by the above construction are identical.

PROOF. An m-partition coloring of $\mathsf{cst}(s\$)$ is completely specified by the colorings on all tree nodes k. Since $\mathsf{path}(k)$ is a substring of s for any node k, π_1 and π_2 will take identical values as $\pi_1 =_s \pi_2$. □

On the other hand, if we are given a m-partition coloring $\tilde{\pi}$ of $\mathsf{cst}(s\$)$ we can canonically construct a m-partition function π by putting

$$\pi(w) \ = \ \begin{cases} \tilde{\pi}(\mathsf{ref}(w)) & \forall w \sqsubseteq s\$ \\ 0 \quad \text{else} \end{cases} \tag{14.4}$$

which allows us to prove the

LEMMA 14.3. π constructed as in equation 14.4 is a m-partition function which fulfills the conditions 14.1 and 14.2.

PROOF. π is defined on Σ^* and surjective since $\tilde{\pi}$ is surjective by definition 14.1(1). Condition 14.1 follows since $\tilde{\pi}(l) > 0$ for all leaves in $\mathsf{cst}(s\$)$ by definition 14.1(2) and condition 14.2 follows because if $\tilde{\pi}$ is surjective on inner nodes, it is also surjective on leaves (via 14.1(3)) and hence π is surjective on all suffixes of $s\$$. □

We can summarize all previous results by

LEMMA 14.4. Given $s \in \Sigma^*$. Then there is a unique bijection between

$$\{[\pi]_{s\$} \ : \ \pi \in \Pi_m(\Sigma \cup \{\$\}) \text{ and } \pi \text{ fulfills conditions 14.1 and 14.2}\}$$

and

$$\{\tilde{\pi} \ : \ \tilde{\pi} \text{ is a } m\text{-partition coloring of } \mathsf{cst}(s\$)\}$$

PROOF. Observe that if π fulfills conditions 14.1 and 14.2 with respect to $s\$$, then every $\pi' \in [\pi]_{s\$}$ fulfills the conditions also. Then the result follows by applying the previously proved lemmata. □

The practical significance of these results is, that we are now able to represent a fixed string and all partitionings distinguishable with respect to the string in one and the same data structure, namely a partition colored suffix tree. The partition coloring π need not be stored separately from the suffix tree but can be stored as node attributes directly in the suffix tree.

14.3. Complexity of Partitionings

Another potential use of representing the equivalence classes of partitionings with respect to s as partition colorings of the suffix tree of $s\$$ is that it suggests a natural measure of *complexity* for partitionings. Why would we want to measure the "complexity" of partitionings? Because it might be relevant in the setting of learning OOMs from finite data.

In statistical learning theory ([**Vap98**], [**Vap99**]) which is all about learning from *finite* data, the complexity of a model class of candidate models is of central importance in controlling the generalization capabilities of an estimated model.

When learning OOMs, we may ask what else besides the OOM dimension contributes to the complexity of the candidate class of OOMs considered when estimating an OOM from finite data. The answer could be that the complexity of indicative and characteristic events used in the estimation might influence the stability of the estimate and the generalization capabilities of the estimated OOM.

How can we define the complexity of a partitioning then? I suggest the following definition might be useful.

DEFINITION 14.2. Given a string $s \in \Sigma^*$ and a *m-partition coloring* π of $T = \mathsf{cst}(s\$)$.

partition complexity Then the *partition complexity* of π is given by

$$C(T) \ := \ \frac{\#\{k \in \mathsf{Nodes}(T) \mid \pi(k) = 0\}}{\#\mathsf{Nodes}(T)}$$

partition border complexity and the *partition border complexity* of π is given by

$$C_B(T) \ := \ \frac{\#\{k \in \mathsf{Nodes}(T) \mid \pi(k) > 0 \ \wedge \ \pi(\mathsf{Parent}(k)) = 0\}}{\#\mathsf{Leaves}(T)}$$

Two examples which illustrate partitionings of minimal and maximal complexity in the sense just introduced are given in figures 14.3 and 14.4. Also, the introduced complexities are well normalized:

PROPOSITION 14.3. For the partition complexity and the partition border complexity it holds that

$$0 \;<\; C(\cdot), C_B(\cdot) \;\leq\; 1$$

PROOF. Observe that a most complex partition coloring of a suffix tree is given when only leaves are colored non-neutrally (> 0). Further a suffix tree with n leaves has at most n inner nodes. This proves the upper bound of 1 which can be reached. The lower bound of 0 follows from the fact that for a partition coloring to be valid, it must be surjective in the colors > 0. □

Intuitively, it seems clear that one should get more stable OOM estimates when using indicative and characteristic events of lower complexity. Of course we can not choose partitionings of arbitrarily low complexity for indicative and characteristic events, since either the partitionings simply will no longer be indicative and characteristic events by violating the non-singularity condition in definition 4.1 or do not longer fully exploit the information contained in the sample.

Due to space and time constraints, I cannot go in to greater detail here. However, one last critical comment: the complexity of indicative and characteristic events seems important, despite the fact that *after* learning the OOM, the indicative and characterstic events used in the estimation are no longer needed. In other words, the final estimated OOM is defined exclusively through it's operators and obviously it's complexity cannot be related to the complexity of any partitionings. However, *during* the process of estimating the OOM we made use of indicative and characteristic events, whose complexity likely effects the stability of the model estimate. How this relates to the generalization capability of the final OOM estimate is another question of course.

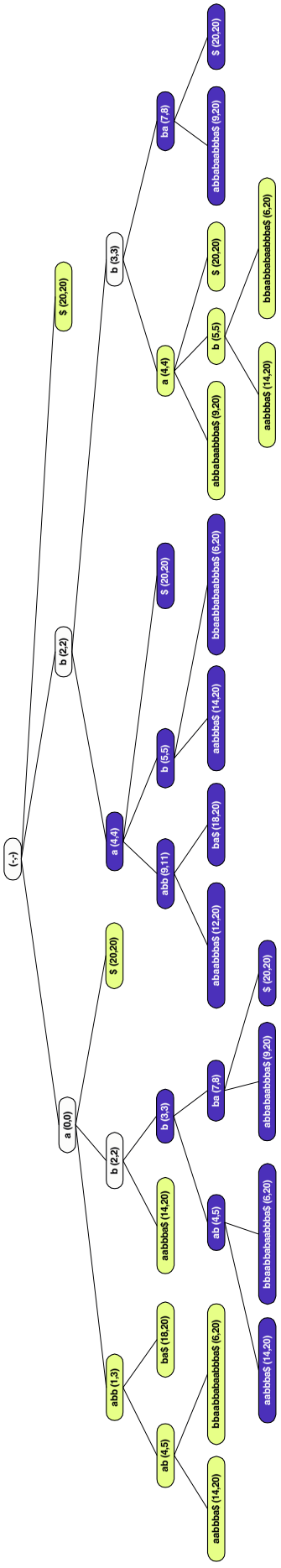

FIGURE 14.1. Example 2-partition coloring of compact suffix tree of *aabbabbbaabbabaabbba*\$.

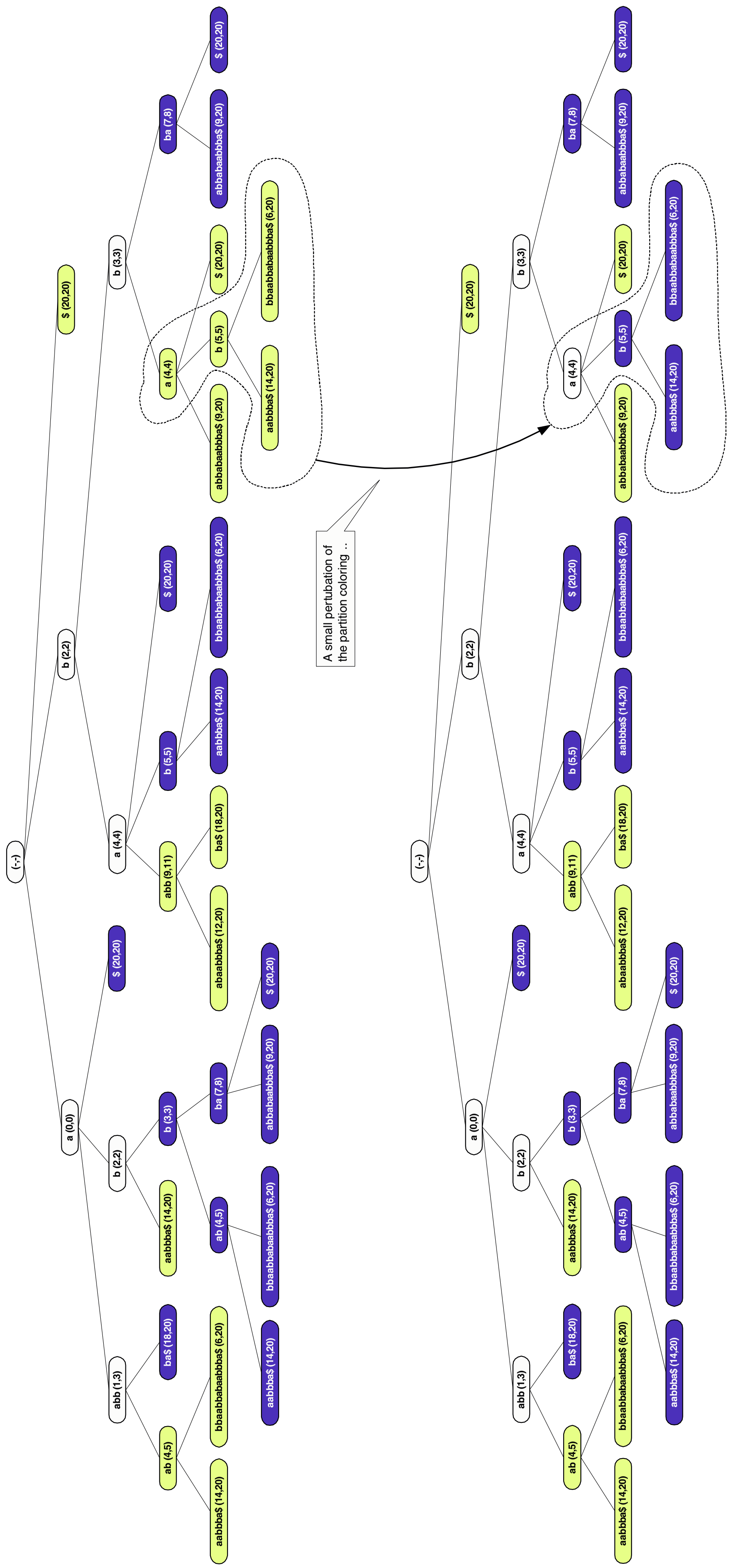

FIGURE 14.2. Example of modification to a 2-partition coloring of compact suffix tree of *aabbabbbaabbabaabbba*\$.

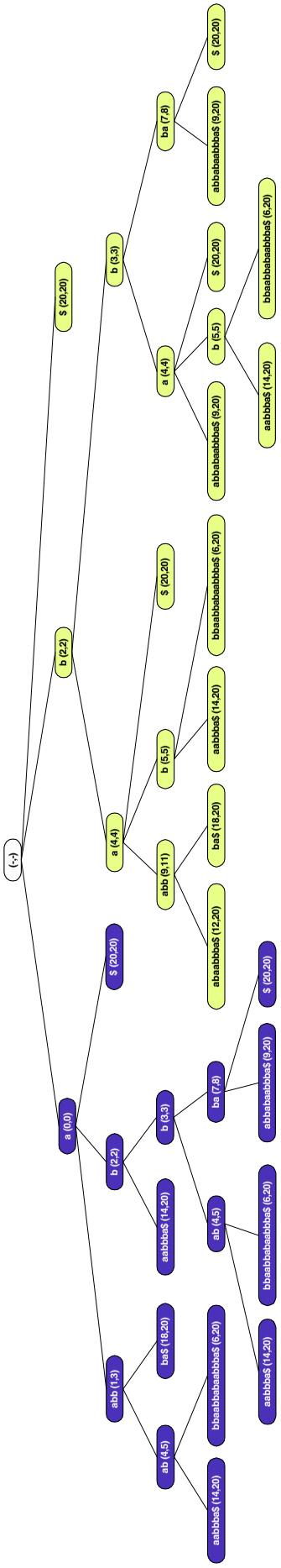

FIGURE 14.3. Example minimal complex 2-partition coloring of compact suffix tree of *aabbabbbaabbabaabbba*\$.

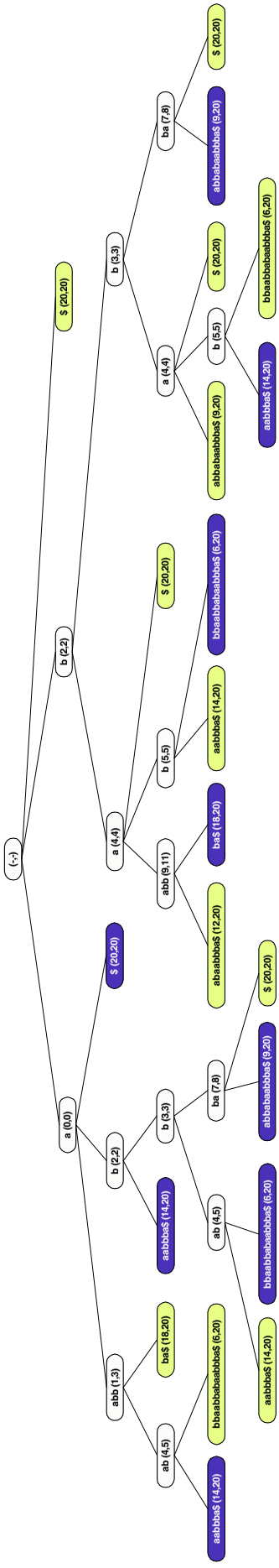

FIGURE 14.4. Example maximal complex 2-partition coloring of compact suffix tree of *aabbabbbaabbabaabbba*\$.

CHAPTER 15

Context Graphs

This section extends concepts introduced in previous sections, in particular suffix trees and partition colorings of suffix trees. I will introduce *context graphs*, which are new data structures developed in this thesis. Context graphs are indexing all past and future contexts within a string. Finally, partition colorings of context graphs are discussed.

With partition colored context graphs we finally have reached one of the goals initially set out in this thesis - to devise a data structure capable of representing both the training sample used in learning an OOM and all candidates of indicative and characteristic events.

15.1. Past and Future Contexts

Given a sample $s \in \Sigma^n$ lets look at the string $\$s\$$, which has s expanded at both ends with a sentinel character $\$ \notin \Sigma$. Obviously, for every $t \in \{0, \ldots, n\}$ the string $\$s\$$ resolves into a prefix and a suffix part

$$\$s\$ \;=\; \$s[1:t]s[t+1:n]\$$$

past context *future context*

which is illustrated in figure 15.1. We will call the prefix $\$s[1:t]$ the *past context* and the suffix $s[t+1:n]\$$ the *future context* to the present t within s.

We have already seen that every suffix (and only the suffixes) of $s\$$, that is every future context within $s\$$ is represented by a leaf in the compact suffix tree $\mathsf{cst}(s\$)$. It is natural to ask where we can find the prefixes of s, that is the past contexts within s.

In section 13.4 we mentioned that the reversed prefixes of a string s are represented by the compact reverse prefix tree $\mathsf{cpt}(s)$ and that this tree is given by

$$\mathsf{cpt}(s) \;=\; \mathsf{cst}(s^{-1})$$

The path labels that can be found in $\mathsf{cpt}(s)$ are all prefixes of s, but *reversed*. This should be clear, since the reverse prefix tree is given as the suffix tree of the *reversed* string s.

Again, since we want to find every (reversed) prefix of s (and only those) as a leaf in the tree, we must ensure that no prefix is nested in s. For this, observe that

$$\mathsf{cpt}(\$s) \;=\; \mathsf{cst}((\$s)^{-1}) \;=\; \mathsf{cst}(s^{-1}\$)$$

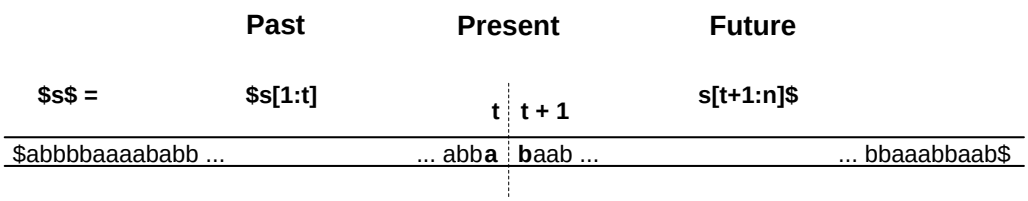

FIGURE 15.1. The past and future context to the present t within s.

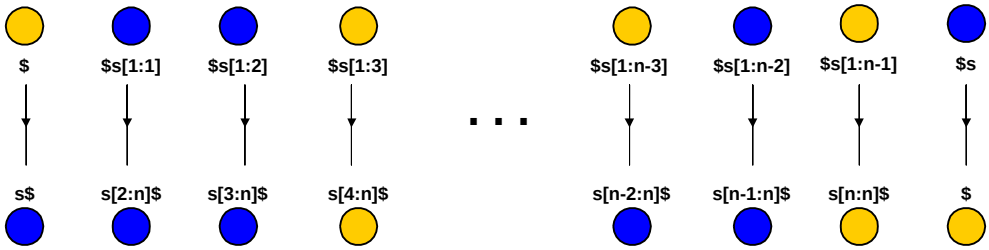

FIGURE 15.2. A partitioning of past and future contexts.

and hence, similarily to appending a sentinel \$ when building the tree for future contexts, we simply *prepend* the sentinel \$ to s in building the tree for past contexts of s. This will do the job since $s^{-1}\$$ has no suffix nested.

In summary, all the past and future contexts of s can be represented as the leaves of the compact reverse prefix tree $\mathsf{cpt}(\$s)$ and the compact suffix tree $\mathsf{cst}(s\$)$ respectively. Of course, those trees will represent and index all reverse and forward substrings of s too.

15.2. Context Partitionings

In chapter 14 we saw how to (uniquely) represent each class of partition functions equivalent with respect to a fixed string $s \in \Sigma^n$ by a particular coloring of the suffix tree $\mathsf{cst}(s\$)$. It was also shown, that it suffices to define the coloring of the leaves of $\mathsf{cpt}(s\$)$. The last section showed that all the past and future contexts of s can be represented as the leaves of $\mathsf{cpt}(\$s)$ and $\mathsf{cst}(s\$)$. Now, this section discusses the results from bringing both insights together.

The net effect in choosing specific indicative and characteristic events is to partition or sort out all past and future contexts actually occurring in a sample s into a number of classes or partitions.

Precisely, we speak of partitions, because those classes are disjunctive and exhaustive with respect to all possible contexts. Nevertheless, in the light of a given, fixed input sample s it is sufficient to define the mapping to partitions for all actual occurring past and future contexts, as for contexts that do not appear in s we may proceed arbitrarily without recognizing any effect.

This is illustrated in figure 15.2, where past and future contexts are partitioned into 2 partitions, blue and yellow, respectively. The upper series shows a possible partition mapping for all $n+1$ past contexts of $\$s\$$, the lower series shows a possible partition mapping for all $n+1$ future contexts of $\$s\$$.

For each present $t \in \{0, \ldots, n\}$ the partitioning determines into which indicative event B_j the past context $\$s[1:t]$ is mapped and into characteristic event A_i the future context $s[t+1, n]\$$ is mapped.

Again, as we saw in chapter 14, the coloring of leaves of the trees, or the partitioning of all past and future contexts already suffice to define the partition colorings on the complete trees.

Yet, the construction of the counting matrices V and $W_x, x \in \Sigma$ needed as the basis for the basic OOM learning algorithm is even possible without any further data structures at all:

(1) For V start with a $m \times m$ matrix with all zeroes and increment V_{ij} each time you observe a pair $(j, i)_t$ when sweeping through the context partitioning as in figure 15.2.

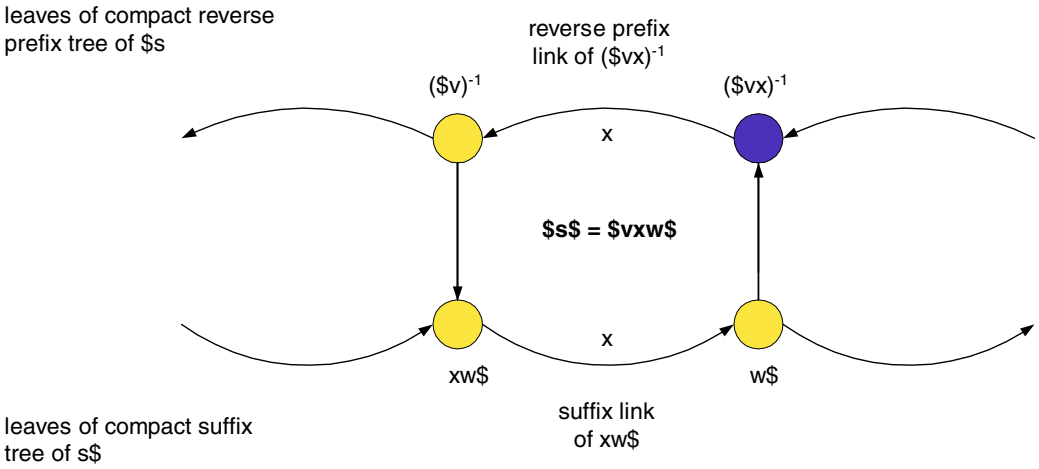

FIGURE 15.3. Commuting suffix links.

(2) For each W_x start with a $m \times m$ matrix with all zeroes and increment $(W_x)_{ij}$ each time you observe an occurence of $(j, \cdot)_t$, $(\cdot, i)_{t+1}$ where $s[t : t] = x$.

How can we observe $(j, \cdot)_t$, $(\cdot, i)_{t+1}$? To see this, recall that in a compact suffix tree of $s\$$, where no suffixes are nested, it holds that

(1) all suffix links are atomic (proposition 13.1)
(2) suffix links of leaves again point to leaves (proposition 13.2)

The situation is illustrated in figure 15.3. We can procede as follows. Given we observe node in the reverse prefix tree with path label $\$v$ (upper left in the figure) at time t. We read off the partition color j (yellow) to determine that $\$v$ will be mapped into the indicative event B_j. We then look at the suffix link of the node in the suffix tree with path label $xw\$$ (lower left) and follow it's suffix link. The suffix link is labeled with the symbol x and leads us to the node with path label $w\$$. Here we read off the partition color i (yellow) to determine that $w\$$ will be mapped into the characteristic event A_i. In summary, we will increment $(W_x)_{ij}$ by one.

Of course this naive procedure is not very satisfying (we discuss this later in the chapter), but illuminates an important point: One may always find indicative and characteristic events such that for instance the V counting matrix obtained from applying the chosen events to the sample s has certain special properties, like minimal condition. In other words and more formally,

LEMMA 15.1. Given a string $s \in \Sigma^n$, $m \in \mathbb{N}$ where $m \geq 2$ and a $m \times m$ matrix V such that

$$(1)\ \sum_{i,j} V_{ij} = n,\ V_{ij} \in \mathbb{N}_0$$
$$(2)\ V \text{ is non-singular}$$

Then there exist indicative events $(B_j)_j$ and characteristic events $(A_i)_i$ that, when applied to s give rise to a counting matrix $(\#B_j A_i)_{ij}$ identical to V.

PROOF. The proof is by construction. First, order all prefixes and suffixes of $\$s\$$ (where $\$ \notin \Sigma$) into a sequence of pairs $(v_t, w_t)_t$, $t \in \{1, \ldots, n+2\}$ as in figure 15.2. Then, color the first V_{11} pairs with the colors $(1, 1)$, the following V_{12} pairs with the colors $(1, 2)$

and so on until coloring the last V_{mm} pairs with colors (m, m). Now define indicative and characteristic events as two partitionings of $(\Sigma \cup \{\$\})^{n+2}$ by putting

$$B_j := \{r \in \Sigma^{n+2} \mid \exists t : v_t \text{ is colored } j \text{ and } v_t \text{ is a suffix of } r\}$$
$$A_i := \{r \in \Sigma^{n+2} \mid \exists t : w_t \text{ is colored } i \text{ and } w_t \text{ is a prefix of } r\}$$

Then $\#B_j A_i$ will be given as the number of prefix-suffix pairs (v_t, w_t) that are colored (j, i). □

Lemma 15.1 shows that any V matrix which fulfills the conditions 15.1(1) and 15.1(2) can be obtained. In particular, this includes V matrices with minimal condition (near 1) and maximal mutual information (see section 6.4). The lemma also indicates, that this is to the expense of a possible huge partitioning space $(\Sigma \cup \{\$\})^{n+2}$ which is exponential in the length of the string s. Obviously, the complexity of the partitionings *must* be involved and considered somehow.

An intuitive argument is this. If we choose arbitrary complex partitionings we will overinterpret the sample. Details in the sample contained only by chance will be given significant effect on our model estimate. In other words, we are overfitting the data. More radically, given a finite sample, the most precise model of the sample is the sample itself.

The general situation of model learning from finite data while avoiding overfitting has been analyzed deeply in *Statistical Learning Theory* ([**Vap98**], [**Vap99**]). There, the key instrument in controlling the problem of overfitting and dually achieving good generalization performance is *model complexity* (VC-dimension or VC-capacity). A somewhat comparable approach is model estimation under the principle of *Minimum Description Length (MDL)* ([**BRY98**]).

Regarding the complexity of indicative and characteristic events, two measures of complexity were already introduced in section 14.3. Those were developed in the present thesis but due to space and time constraints a thorough analysis of the complexity measures could not be conducted.

15.3. Context Graphs

Context graphs are special directed graphs. A introduction to the general theory of graphs is [**Die96**]. Context graphs were developed in this thesis from insights into the relation of reverse prefix and suffix trees of a string $s \in \Sigma^*$ and how they represent all past and future contexts within $\$s\$$. Context graphs combine the nodes and edges from the compact reverse prefix tree and the compact suffix tree of the input string. Further, context graphs have additional edges which connect the nodes of both trees, the *context links*.

Using context links, we can quickly give answers to questions like given a *partial future context* $w \in \Sigma^*$ give me all past contexts $\$v_j \in \Sigma^*$ after which w occurred in s, that is incidences where $\$v_j w$ is a prefix of $\$s\$$. Note that a full future context $w\$$ defines a unique position in s since it ends with the sentinel \$ which nowhere else appears in s, whereas a partial future context can appear at different positions in s. Or, given a *partial past context* $v \in \Sigma^*$ give me all future contexts $w_i\$$ before which v occurred, that is incidences where $vw_i\$$ is a suffix of $\$s\$$. Again, a partial past context can appear more than once in s.

partial future context

partial past context

I will introduce context graphs formally and then discuss these aspects in more detail. Note that we introduced $\mathsf{cpt}(\$s)$ to denote the compact reverse prefix tree of $\$s$ and $\mathsf{cst}(s\$)$ to denote the compact suffix tree of $s\$$.

Definition 15.1. Given a string $s \in \Sigma^*$ and a sentinel $\$ \notin \Sigma$ the *context graph* of s

context graph

denoted $\mathsf{ctg}(s)$ is the directed graph with nodes

$$\mathsf{Nodes}(\mathsf{ctg}(s)) \;:=\; \mathsf{Nodes}(\mathsf{cpt}(\$s)) \;\cup\; \mathsf{Nodes}(\mathsf{cst}(s\$))$$

where each node k has three outgoing edges given by the target nodes $\mathsf{parent}(k)$, $\mathsf{suffix}(k)$ and $\mathsf{context}(k)$ the edges project to and defined by

$$\mathsf{parent}(k) \;:=\; \text{parent node of } k \text{ in either } \mathsf{cpt}(\$s) \text{ or } \mathsf{cst}(s\$)$$
$$\mathsf{suffix}(k) \;:=\; \text{target node of suffix link of } k \text{ in either } \mathsf{cpt}(\$s) \text{ or } \mathsf{cst}(s\$)$$

and let $\mathsf{context}(k) \in \mathsf{Nodes}(\mathsf{ctg}(s))$ recursively defined by

$$\mathsf{path}(k)^{-1}\mathsf{path}(\mathsf{context}(k)) = \$s\$ \quad \text{if } k \in \mathsf{Leaves}(\mathsf{cpt}(\$s))$$
$$\mathsf{path}(\mathsf{context}(k))^{-1}\mathsf{path}(k) = \$s\$ \quad \text{if } k \in \mathsf{Leaves}(\mathsf{cst}(s\$))$$

and

$$\mathsf{context}(k) \;:=\; \mathsf{LCA}(\{\mathsf{context}(h) \mid h \in \mathsf{Leaves}(k)\})$$

if k is not a leaf.

least common ancestor

Here, LCA denotes the *least common ancestor* node of a given non-empty set of nodes. The reader may easily verify that in above definition, the set of nodes LCA is applied to is never empty, hence l is always well defined. Also note, that we agree on $\mathsf{parent}(root) = root$ in both trees.

A schematic drawing of a context graph with some context links is given in figure 15.4. A detailed example for a context graph is given in figure 15.3.

As may be seen from the definition and suggested in figure 15.3, the context links constitute a bijection between the *leaves* of the reverse prefix and suffix tree.

For an *inner* node k, the context link of k points to node l in the "opposite" tree such that l is the least common ancestor of the leaves that are pointed to by context links of the leaves under k. Context links of inner nodes are further illustrated in the figures 15.3, 15.3, 15.3 and 15.3.

For convenience, the following definition declares names for the two node sets making up the nodes of a context graph.

DEFINITION 15.2. Given a context graph $\mathsf{ctg}(s)$ we will denote the nodes of the reverse prefix tree part as

$$\mathsf{PrefixNodes}(\mathsf{ctg}(s)) := \mathsf{Nodes}(\mathsf{cpt}(\$s))$$

and the nodes of the suffix tree part as

$$\mathsf{SuffixNodes}(\mathsf{ctg}(s)) := \mathsf{Nodes}(\mathsf{cst}(s\$))$$

The context links give us direct access to all forward and reverse continuations of arbitrary substrings in s. This was already said in the introduction to this section and we are now able to state and prove this claim formally.

LEMMA 15.2. Given $s \in \Sigma^*$, a substring $t \sqsubseteq s$ and the context graph $\mathsf{ctg}(s)$ of s. Let $\mathsf{ref}_{\mathrm{Prefix}}(t^{-1})$ denote the reference node of t^{-1} within the reverse prefix tree part of $\mathsf{ctg}(s)$ and $\mathsf{ref}_{\mathrm{Suffix}}(t)$ denote the reference node of t within the suffix tree part of $\mathsf{ctg}(s)$. Then all forward continuations $tw \sqsubseteq s$ of t in s may be found in the subtree hanging off the node

$$\mathsf{context}(\mathsf{ref}_{\mathrm{Prefix}}(t^{-1}))$$

and all backward continuations $wt \sqsubseteq s$ of t in s may be found in the substree hanging off node

$$\mathsf{context}(\mathsf{ref}_{\mathrm{Suffix}}(t))$$

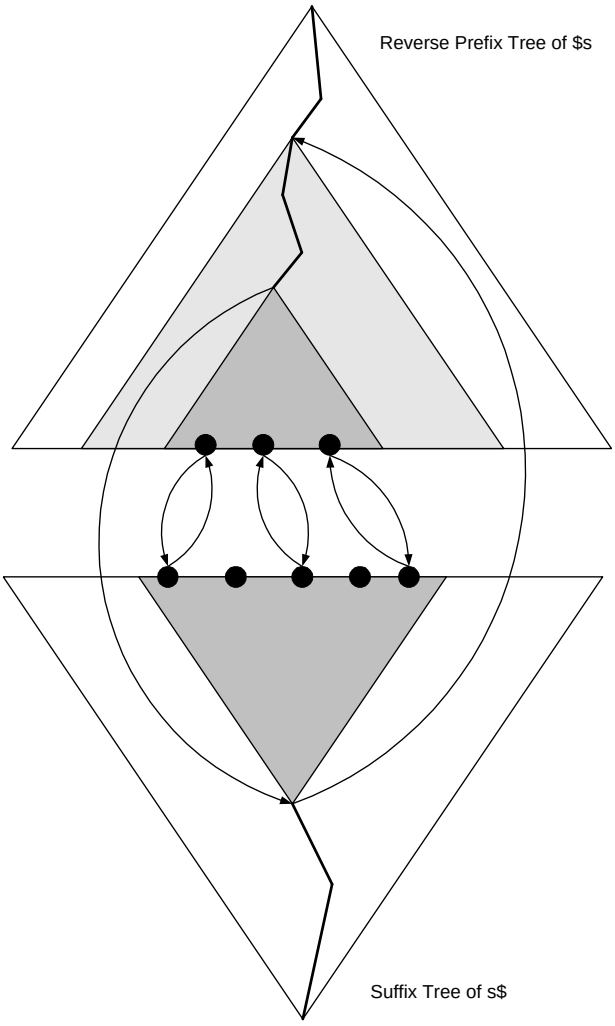

FIGURE 15.4. Schematic drawing of a context graph with some context links.

PROOF. If $k := \mathsf{ref}_{\mathrm{Prefix}}(t^{-1})$ is a leaf in the reverse prefix tree part of $\mathsf{ctg}(s)$ then t occurs only once in s, $\mathsf{path}(k)^{-1}\mathsf{path}(\mathsf{context}(k)) = \$s\$$ by definition of context links, $\mathsf{context}(k)$ is a leaf in the suffix tree part of $\mathsf{ctg}(s)$ and all continuations of t in s can be found as prefixes of $\mathsf{path}(\mathsf{context}(k))$. If k is a non-leaf inner node then every occurrence of t in s is represented by a leaf $h \in \mathsf{Leaves}(k)$ and all continuations of t in s can be found as prefixes of leaves $\mathsf{path}(\mathsf{context}(h))$ in the suffix tree part of $\mathsf{ctg}(s)$. But then those continuations will also be found in the subtree hanging off the least common ancestor node $\mathsf{LCA}(\{\mathsf{context}(h) \mid h \in \mathsf{Leaves}(k)\}$ which is the target node $\mathsf{context}(k)$ by definition of context links. The proof for backward continuations is similar. □

The structure of context graphs has three aspects. First, the parent edges constitute the involved reverse prefix and suffix trees. Second, the suffix edges make up the suffix links of the involved trees. Up to this point, the structure is nothing more than the union of the structures of both trees. On the other hand, the context edges and the context links they represent impose a third and different level of structure onto the set of nodes. The following proposition will shed some light onto the structure thus added and should be compared to figure 15.4.

PROPOSITION 15.1. Given $s \in \Sigma^*$ and the context graph $\mathsf{ctg}(s)$ of s. Then for all $k \in \mathsf{Nodes}(\mathsf{ctg}(s))$ it holds that

(1) $k \in \mathsf{Descs}(\mathsf{context}(\mathsf{context}(k)))$
(2) $\mathsf{context}(k) \in \mathsf{Descs}(\mathsf{context}(\mathsf{Parent}(k)))$

PROOF. In general, given subsets of leaf nodes $L_1, L_2 \subset \mathsf{Leaves}(T)$ of some tree T. Then $L_1 \subset L_2$ implies $\mathsf{LCA}(L_1) \in \mathsf{Descs}(\mathsf{LCA}(L_2))$. The proposition is now easy to see from the definition of context links in context graphs. □

The last notion I introduce and discuss was motivated by the following observation. Suppose $k = \mathsf{context}(\mathsf{context}(k))$. What does this mean? It means that every occurrence of the (partial) past or future context represented by node k is *always* co-occurring with the (partial) future or past context $\mathsf{context}(k)$. But if this is the case, this relation will also hold in the opposite direction. What we have is a special case of proposition 15.1(1). Now, with OOM learning and choosing good indicative and characteristic events, we are usually indeed interested in the *co-occurrence* of partial past and future contexts as candidates for or taking part in indicative and characteristic events. This motivated the following

context matching factor

DEFINITION 15.3. Given $s \in \Sigma^*$ and the context graph $\mathsf{ctg}(s)$ of s. Then the *context matching factor* of a node $k \in \mathsf{Leaves}(\mathsf{ctg}(s))$ is given by

$$\frac{\#\mathsf{Leaves}(k)}{\#\mathsf{Leaves}(\mathsf{context}(k))}$$

The *context matching factor* can be seen as measuring the co-occurrence of the context represented by node k and the context represented by node $\mathsf{context}(k)$. The factor has a number of trivial properties

PROPOSITION 15.2. For any node $k \in \mathsf{Leaves}(\mathsf{ctg}(s))$ it holds

(1) $g(k) = 1 \quad \forall k \in \mathsf{Leaves}(\mathsf{ctg}(s))$
(2) $g(k) = 1 \quad \text{if } k = root$
(3) $0 < g(k) \leq 1$

PROOF. (1) holds since $\mathsf{context}(k) \in \mathsf{Leaves}(\mathsf{ctg}(s))$ for all leaves $k \in \mathsf{Leaves}(\mathsf{ctg}(s)$. (2) follows from the fact that $\#\mathsf{Leaves}(\mathsf{cpt}(\$s)) = \#\mathsf{Leaves}(\mathsf{cst}(s\$))$ and $\mathsf{context}(root_{\mathsf{cpt}(\$s)}) = root_{\mathsf{cst}(s\$)}$ and $\mathsf{context}(root_{\mathsf{cst}(s\$)}) = root_{\mathsf{cpt}(\$s)}$. (3) follows from (1) and the fact that $\#\mathsf{Leaves}(\mathsf{LCA}(L)) \geq \#L$ for every subset L of leaves in a tree. □

15.4. Partition Colored Context Graphs

In this and the last chapter it was discussed how colorings of the reverse prefix and the suffix tree of a string can define partitionings and thereby indicative and characteristic events.

To complete our set of notions, I shortly give a formal definition of *partition colored context graphs* that merely subsumes the notions already introduced.

DEFINITION 15.4. Given $s \in \Sigma^*$. Then a *partition colored context graph* of s is a context graph $\mathsf{ctg}(s)$ of s where the reverse prefix tree and suffix tree parts of $\mathsf{ctg}(s)$ have a valid partition coloring (as in definition 14.1).

A schematic drawing of a partition colored context graph is given in figure 15.11. Note, that in this drawing the leaves of both trees have been reordered, which results from the natural lexicographical order within a suffix tree or reverse prefix tree that is induced from

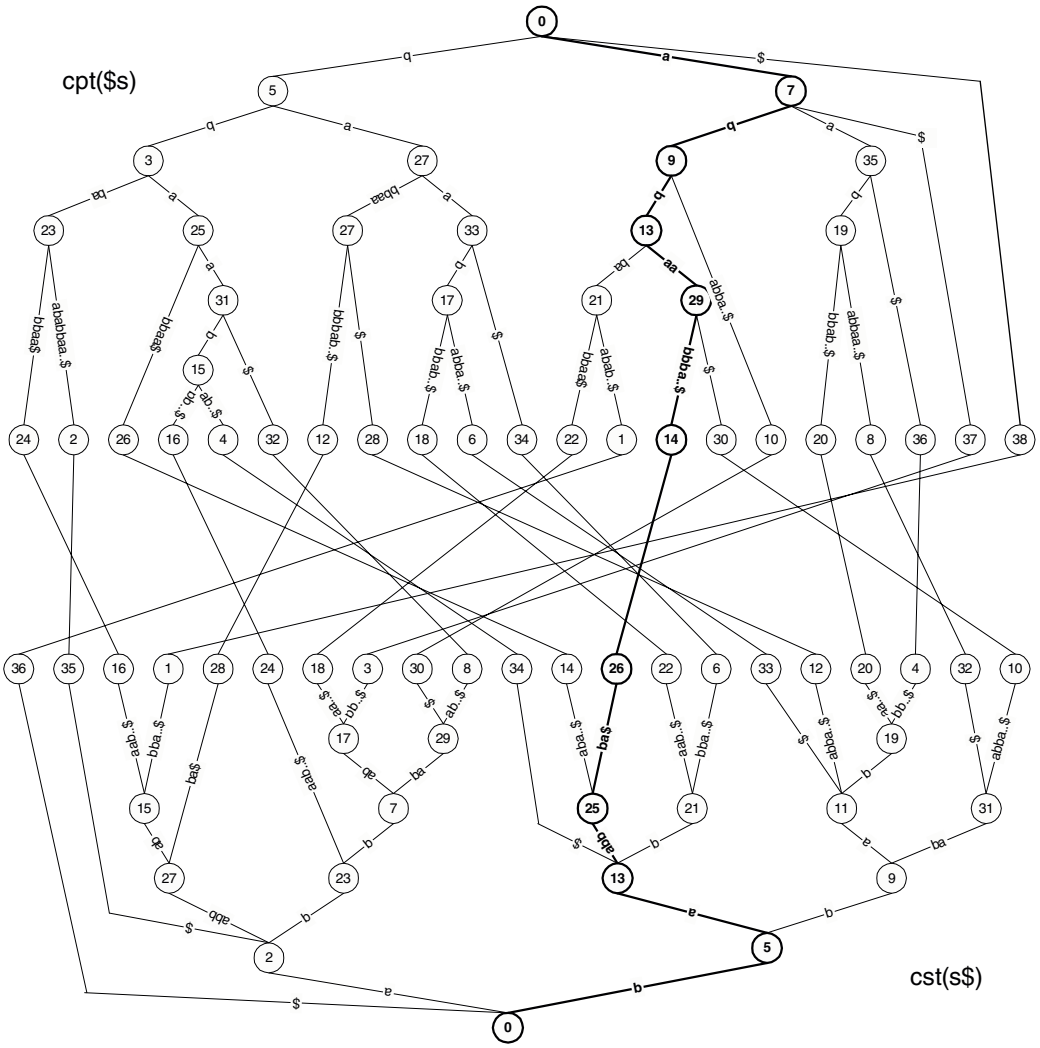

FIGURE 15.5. Compact context graph for $\$s\$ = \$aabbabbbaabbabaabbba\$$. The thick lines represent the reverse prefix $(\$aabbabbbaabba)^{-1}$ (node 14 in $\mathsf{cpt}(\$s)$) and the suffix $baabbba\$$ (node 26 in $\mathsf{cst}(s\$)$). The thick line between nodes 14 and 26 represents the forward and backward context links that link those two nodes. All other context links and all suffix and reverse prefix links are not shown.

an ordering of Σ.

Every path from top to bottom of the context information tree goes through the reverse prefix tree and through the suffix tree and corresponds to some present t. The thick line in figure 15.11 is an example for $t = 2$.

The leaves of both trees have been labeled by the path-labels of the respective nodes. The leaves of the reverse prefix tree have been labeled by $\mathsf{path}(\cdot)^{-1}$ for convenience.

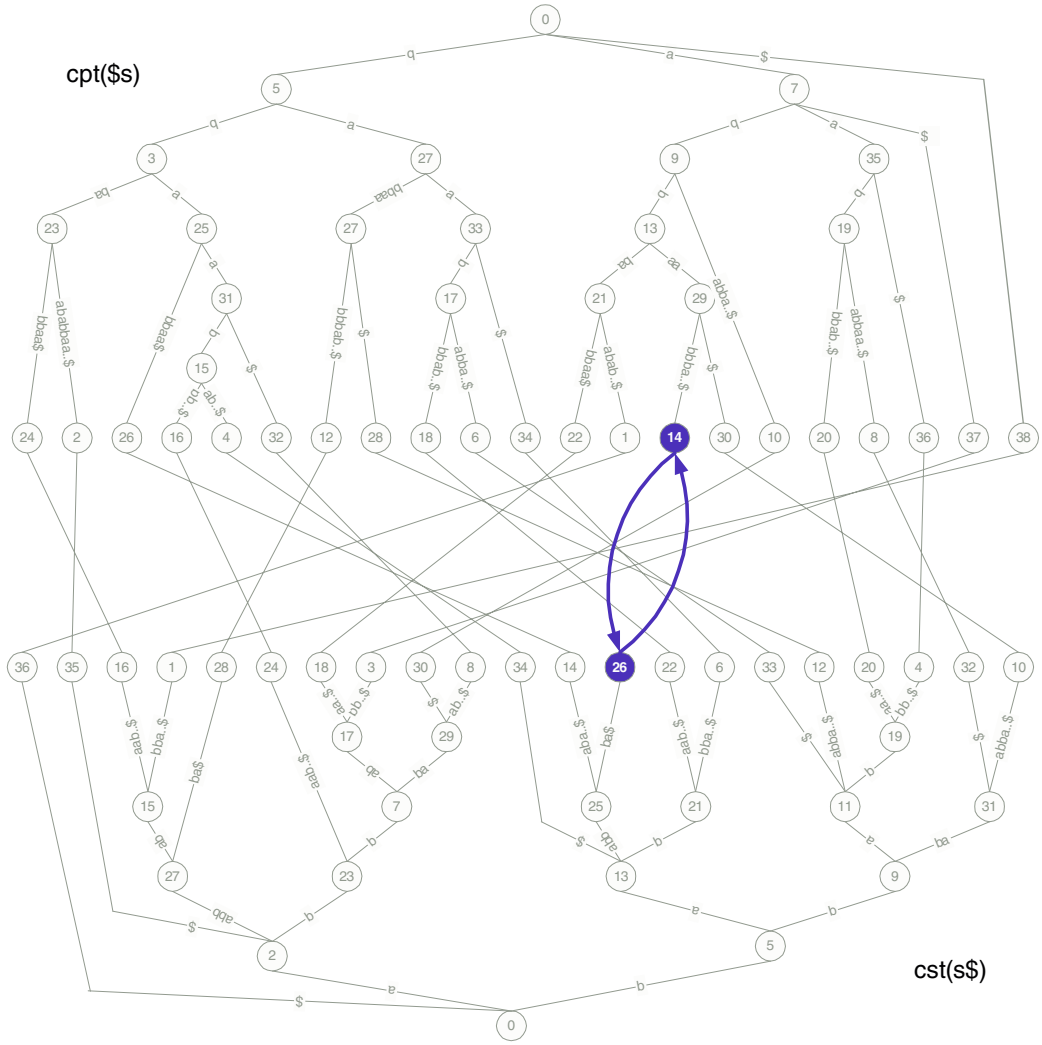

FIGURE 15.6. Compact context graph for $\$s\$ = \$aabbabbbaabbabaabbba\$$. The thick blue lines show the forward and backward context links that link two leaf nodes in the opposite trees.

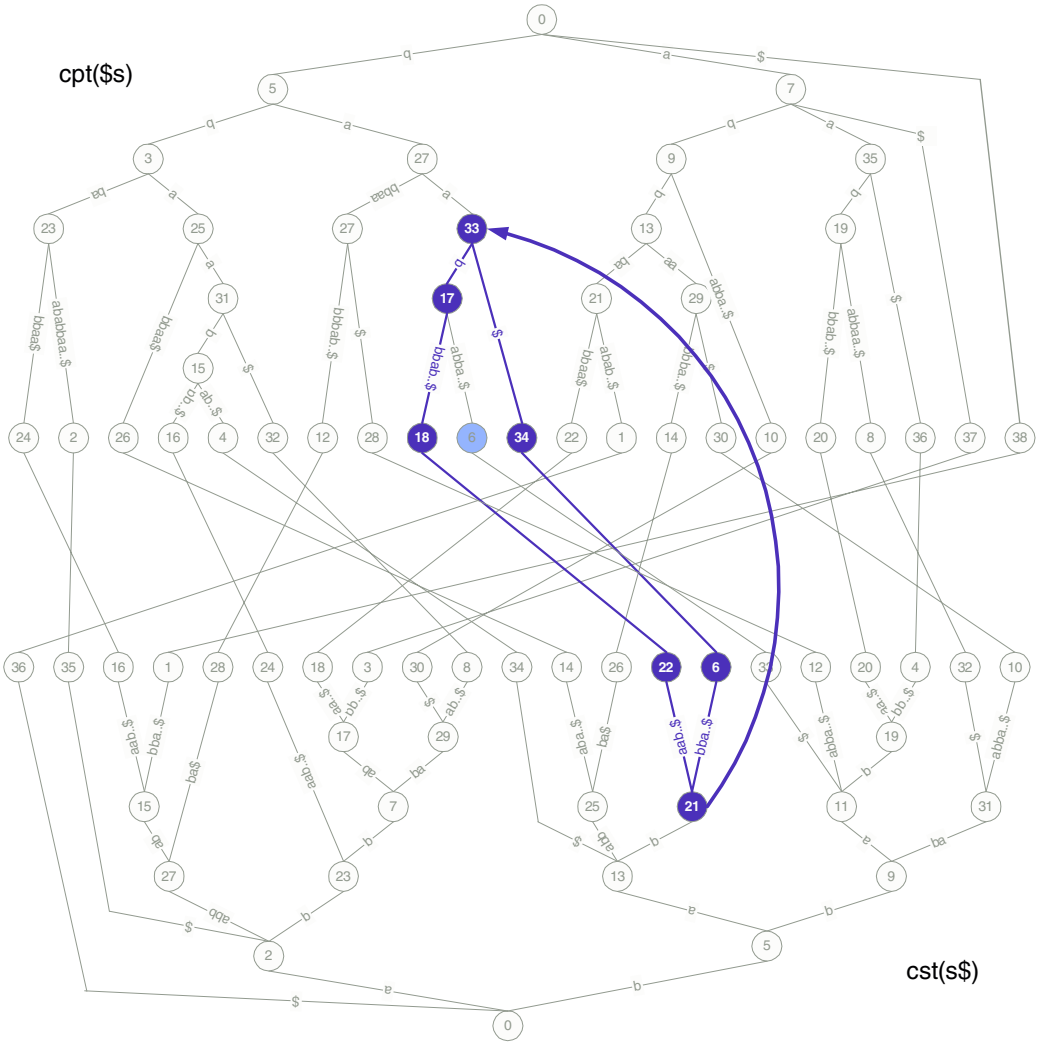

FIGURE 15.7. Compact context graph for $\$s\$ = \$aabbabbbaabbabaabbba\$$. The thick blue twisted line with arrow shows the context link of node 21 in $\mathsf{cst}(s\$)$ representing the substring bab pointing to node 33 in $\mathsf{cpt}(\$s)$ representing the reverse substring $(aab)^{-1}$.

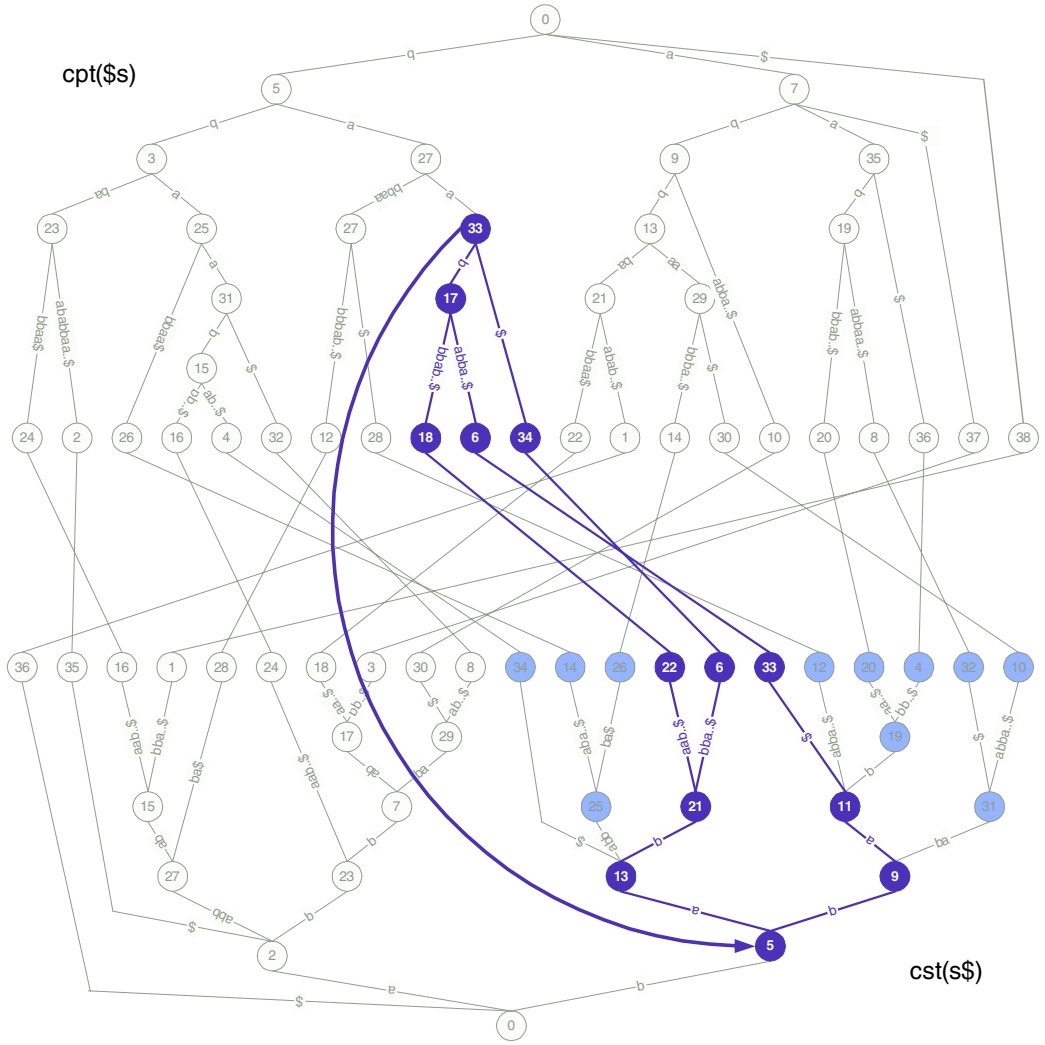

FIGURE 15.8. Compact context graph for $\$s\$ = \$aabbabbbaabbabaabbba\$$. The thick blue twisted line with arrow shows the context link of node 33 in $\mathsf{cpt}(\$s)$ representing the reverse substring $(aab)^{-1}$ pointing to node 5 in $\mathsf{cst}(s\$)$ representing the substring b.

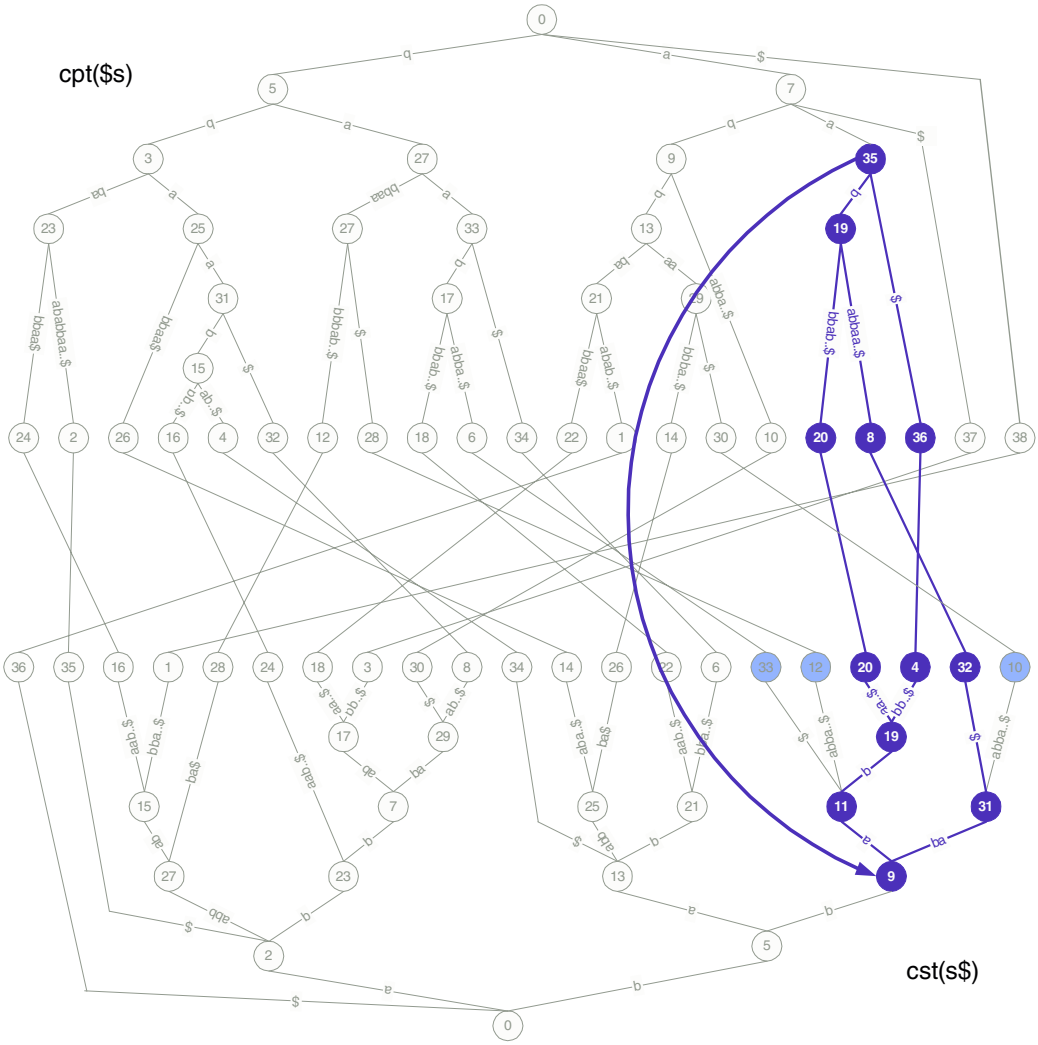

FIGURE 15.9. Compact context graph for $\$s\$ = \$aabbabbbaabbabaabbba\$$. The thick blue twisted line with arrow shows the context link of node 35 in $\mathsf{cpt}(\$s)$ representing the reverse substring $(aa)^{-1}$ pointing to node 9 in $\mathsf{cst}(s\$)$ representing the substring bb.

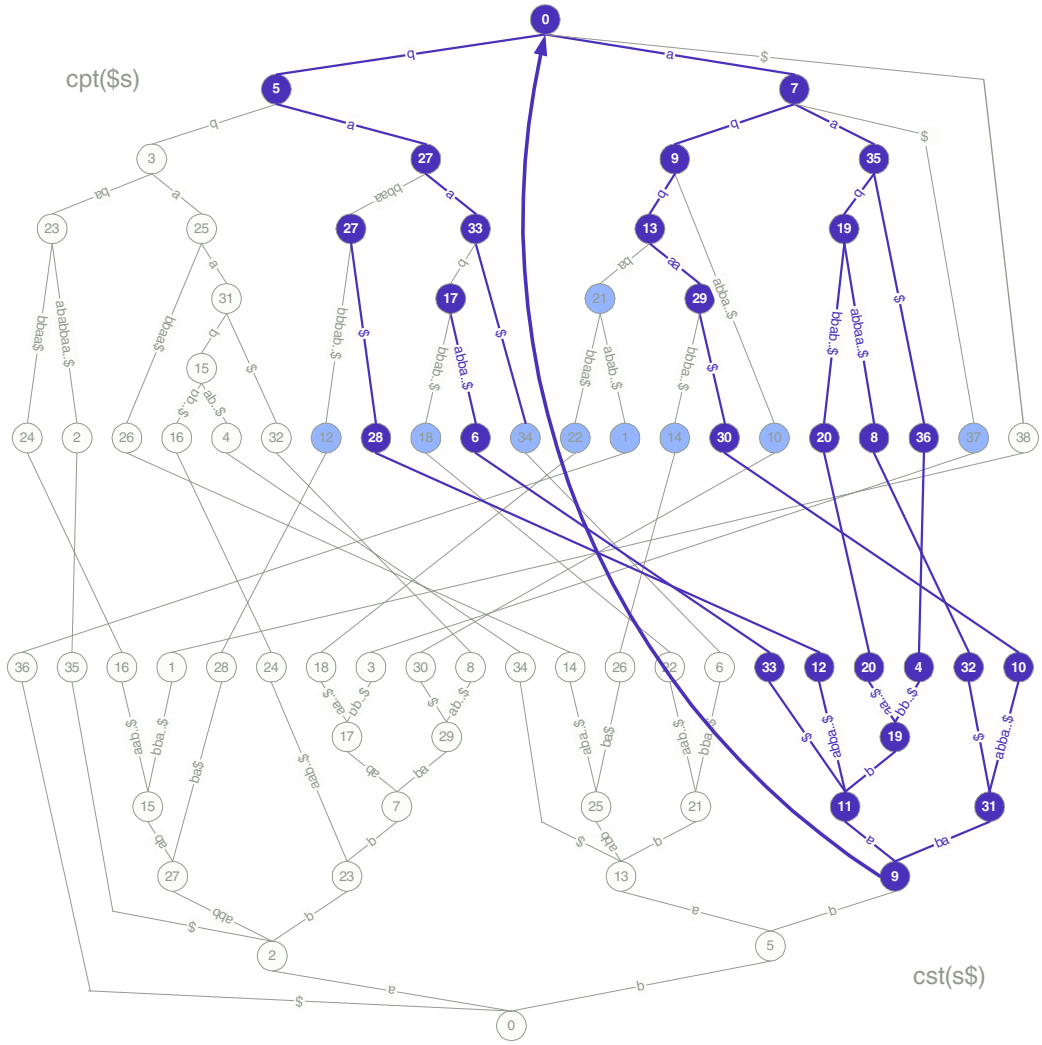

FIGURE 15.10. Compact context graph for $\$s\$ = \$aabbabbbaabbabaabbba\$$. The thick blue twisted line with arrow shows the context link of node 9 in $\mathsf{cst}(s\$)$ representing the substring bb pointing to node 0 in $\mathsf{cpt}(\$s)$ representing the substring ϵ.

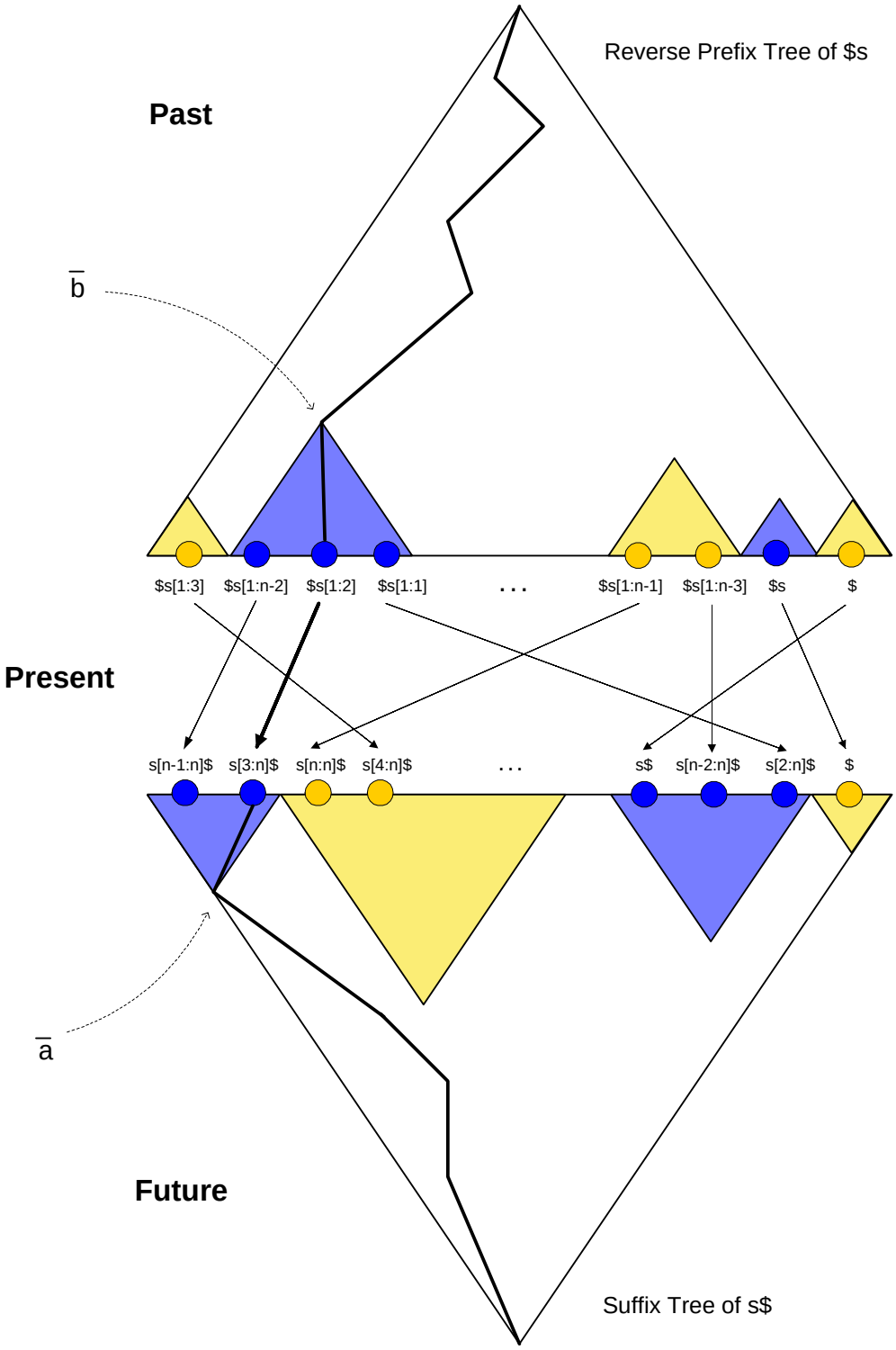

FIGURE 15.11. Schematic drawing of a partition colored context graph.

CHAPTER 16

The COPRUMIC Learning Algorithm

In the last chapter it was shown how to use context graphs and colorings of context graphs in representing strings and characteristic and indicative events. In this chapter, I present a complete heuristic learning algorithm for OOMs that builds on context graphs and the notion of mutual information.

The COPRUMIC (**CO**ntext **PR**uning **M**utual **I**nformation **C**lustering) learning algorithm is a method of OOM estimation from finite data. It is build around two ideas:

(1) context pruning of suffixtrees
(2) clustering based on mutual information

Both ideas and the resulting algorithm are detailed in the following sections. This is done by a top-down decomposition of the COPRUMIC algorithm.

It is worth noting, that the clustering part builds on the context pruning part, but the context pruning method can be deployed independently. In fact, in [**Kre**] another OOM estimation method is built on top of the context pruning algorithm.

16.1. COPRUMIC

The high level structure of COPRUMIC is given below. It works by first construction so-called *raw* counting vectors and matrices for the events A_i, B_j and B_jA_i and B_jxA_i. This is done via the CONTEXT-PRUNING method, which is described in subsection 16.4.

The final form of the counting matrices V and $W[x]$ is computed from the raw counting vectors and matrices by an algorithm called MUTUAL-INFORMATION-CLUSTERING which is described in subsection 16.3. Finally, an OOM estimate is computed by an embedded basic OOM learning algorithm.

COPRUMIC$(s, pdim, dim)$
1 $A_{raw}, B_{raw}, V_{raw}, W_{raw} \leftarrow$ CONTEXT-PRUNING$(s, pdim)$
2 $V_{raw} \leftarrow V_{raw}/|s|$
3 $V, Cl_{rows}, Cl_{cols} \leftarrow$ MUTUAL-INFORMATION-CLUSTERING(V_{raw}, dim)
4 $A \leftarrow$ REDUCE-VECTOR$(A_{raw}, dim, Cl_{rows})$
5 $w_0 \leftarrow 1/|s| \cdot A$
6 **for** $x \in \Sigma$
7 **do**
8 $W[x] \leftarrow$ REDUCE-MATRIX$(W_{raw}[x], dim, Cl_{rows}, Cl_{cols})$
9 $\tau[x] \leftarrow W[x]V^{-1}$
10 **return** $w_0, \tau[]$

The run-time and space complexities of COPRUMIC is clearified by the following

LEMMA 16.1. The COPRUMIC algorithm has a run-time complexity of

$$\mathcal{O}(|s| + (|\Sigma| \cdot pdim)^2 + pdim^4 + |\Sigma| dim^3)$$

and a space complexity of

$$\mathcal{O}(|s| \log pdim + |\Sigma| dim^3)$$

PROOF. The run-time is of order $|\Sigma|dim^3$ because of the matrix multiplications in $W[x]V^{-1}$ for computing the final operator estimates. The run-time is of order $pdim^4$ because V_{raw} has a dimension of order $pdim \times pdim$ and the run-time complexity of the MUTUAL-INFORMATION-CLUSTERING algorithm is then $\mathcal{O}(pdim^4)$. Further, the run-time complexity is of order $|s| + (|\Sigma| \cdot pdim)^2$ as this holds for the CONTEXT-PRUNING algorithm also. The space complexity is of order $|s| \log pdim$ since suffixtrees take order of $|s|$ space and colored suffixtrees must take $\log pdim$ additional storage per node to store the assigned partition. Finally, space complexity is of order $|\Sigma|dim^3$ since this is what it takes to store all operators $\tau[x]$. □

16.2. Reduce-Vector and Reduce-Matrix

The following two functions are fairly trivial helper functions to reduce vectors and matrices by row and column merges specified by merge (reduction) vectors.

The function REDUCE-VECTOR takes a vector $vec[n]$ of dimension n, a target dimension dim and a merge vector $cl[n]$ and reduces vec to a new vector $nvec[dim]$ of dimension dim. The merge vector contains in it's entries the target indices for entries in the original vector. Assume for example $cl[5] = 3$, then $vec[5]$ will be merged (added) to target vector entry $nvec[3]$. Obviously, $n \geq dim$ must hold.

```
REDUCE-VECTOR(vec[n], dim, cl[n])
 1  nvec ← new vector of dimension dim
 2  nvec ← 0
 3  for i = 1 ... n
 4      do
 5          k ← cl(i)
 6          nvec(k) ← nvec(k) + vec(i)
 7  return nvec
```

The function REDUCE-MATRIX takes a matrix $mat[n, m]$ of dimension nxm, a target dimension dim, a row merge vector $clr[n]$ and a column merge vector $clc[n]$ and reduces mat to a new matrix $nmat[dim, dim]$ of dimension $dim \times dim$. The merge vectors contain in their entries the target indices for entries in the original matrix. Assume for example $clr[5] = 3$ and $clc[4] = 1$, then $mat[5, 4]$ will be merged (added) to target matrix entry $nmat[3, 1]$. Obviously, $n \geq dim$ and $m \geq dim$ must hold.

```
REDUCE-MATRIX(mat[n, m], dim, clr[n], clc[m])
 1  nmat ← new matrix of dimension dim × dim
 2  nmat ← 0
 3  for i = 1 ... n
 4      do
 5          for j = 1 ... m
 6              do k ← clr(i)
 7                 l ← clc(j)
 8                 nmat(k, l) ← nmat(k, l) + mat(i, j)
 9
10  return nmat
```

16.3. Mutual-Information-Clustering

The mutual information clustering algorithm is a method of stepwise reducing the dimensions of a $u \times w$ matrix to a quadratic matrix of dimension $d \times d$ by row and column merges. The input matrix must have all positive entries that sum up to 1.

In each reduction step, a single pair, either a row or a column pair is merged by adding them. This is done until a finally dxd matrix is reached. Obviously, $u \geq d$ and $w \geq d$ must hold. The column or row pair to be merged in each step is chosen such that the reduction of mutual information of the intermediate matrix is minimal among all candidate pairs. The mutual information of a matrix was introduced in 6.7.

```
MUTUAL-INFORMATION-CLUSTERING(M, d)
 1  R ← M
 2  clr ← new vector of dimension rows(R)
 3  clc ← new vector of dimension columns(R)
 4  while rows(R) > d or columns(R) > d
 5      do
 6          find pair i, j of row or column indices such that R
 7            with i, j merged has highest mutual information
 8          merge rows or columns i, j
 9          track i, j merge in clr or clc
10  return R, clr, clc
```

The algorithm is a local, greedy cluster method. The function returns the reduced matrix R and two vectors clr and clc that contain information about which rows and columns have been merged during the reduction process. The merge vectors contain in their entries the target indices in the reduced matrix for entries in the original matrix. Assume for example $clr[5] = 3$ and $clc[4] = 1$. Then $M(5, 4)$ was merged (added) to target matrix entry $R(3, 1)$. Obviously, $rows(M) \geq d$ and $columns(M) \geq d$ must hold.

LEMMA 16.2. The algorithm MUTUAL-INFORMATION-CLUSTERING has a run-time given by

$$\mathcal{O}(n^4) \quad \text{where} \quad n = \max\{rows(M), columns(M)\}$$

PROOF. Checking all possible pairs of rows and columns for the resulting mutual information when the matrix would be reduced accordingly takes less than $2n^2$ steps. Further, the matrix can at most be reduced $2n$ times. Each check for the mutual information that would result can be done without recomputing the mutual information of the candidate matrix from scratch but only by adjusting the mutual information from a intermediate matrix by the change introduced by a row or column pair merge. To compute this change, n matrix entries have to be touched. This finishes the proof. □

16.4. Context-Pruning

The context pruning algorithm takes as input a sample $s \in \Sigma^n$ and a dimension $pdim \in \mathbb{N}$, $pdim \geq 2$. The algorithm first builds the suffixtree of $s\$$ and the suffixtree of $(\$s)^{-1}$, the latter of which is the reverse prefix tree of $\$s$. Here, $\$$ is assumed to be a sentinel character not contained in the alphabet Σ.

Next, both trees are partition colored using a suffix pruning algorithm described in the next section. Both trees then have a complete and valid partition coloring. In particular, all the leaves of both trees are assigned a partition, that is indicative and characteristic events.

Finally, the counting vectors for indicative and characteristic events B and A and the containing matrices V and $W[x]$, $\forall x \in \Sigma$ are computed. All vectors and matrices are computed merely from the partition indices assigned to leaf nodes in both trees. That is, assume the leaf $l \in \mathsf{Leaves}(cpt)$ representing a concrete past context has assigned a partition index (indicative event) $j = partition(l)$. Then the concrete future context immediately following the conditioning past context is represented by $contextLink(l)$ and has assigned a partition index (characteristic event) given by $i = partition(contextLink(l))$.

Here, $contextLink(l)$ is the node in cpt obtained by following the context link from l. Note, that in this algorithm we only need context links for leaves. We do not need a complete context graph which would contain context links for all nodes in both trees. The context links for leaves can be computed on the fly (described later). Up to now, we have already seen how to compute A, B and V. What is left is a description how the algorithm computes $W[x]$. The key insight is illustrated in figure 15.3. The figure shows some leaves of the suffix and reverse prefix trees. Also shown are context links (in one direction) and suffix links. The point is that for leaves in a compact suffixtree, all suffix links point to nodes which are again leaves. We can use the situation to count events of the form $B_j x A_i$ like follows.

Assume the leaf $l \in \mathsf{Leaves}(cpt)$ representing a concrete past context has assigned a partition index (indicative event) $j = partition(l)$. Further let $\mathsf{path}(l) = \$b$ be the path of the leaf l. If we now follow the suffix link arriving in l in reversed direction, we reach a leaf $revSuffixLink(l)$ in the reverse prefix tree with path label $\$bx$ for some $x = revSuffixLink(l)[1] \in \Sigma$. Now from there, follow the context link which leads us to a leaf $contextLink(revSuffixLink(l))$ in the suffix tree with some path $a\$$. It then holds that $\$s\$ = \$bxa\$$ and we found an incident of the event

$$B_{partition(l)} x A_{partition(contextLink(revSuffixLink(l)))}$$

```
CONTEXT-PRUNING(s, pdim)
 1  cst ← SUFFIX-TREE(s$)
 2  cpt ← SUFFIX-TREE(($s)^{-1})
 3  cut ← |s$| / pdim
 4  CstBorderNodes, CstPartitionCount ← PRUNE-SUFFIXTREE(cst, cut, 1, root(cst))
 5  CptBorderNodes, CptPartitionCount ← PRUNE-SUFFIXTREE(cpt, cut, 1, root(cpt))
 6  A ← new CstPartitionCount vector
 7  A ← 0
 8  B ← new CptPartitionCount vector
 9  B ← 0
10  V ← new CstPartitionCount x CptPartitionCount matrix
11  V ← 0
12  for x ∈ Σ
13      do
14          W[x] ← new CstPartitionCount x CptPartitionCount matrix
15          W[x] ← 0
16  for l ∈ Leaves(cpt)
17      do
18          i ← partition(contextLink(l))
19          j ← partition(l)
20          A(i) ← +1
21          B(j) ← +1
22          V(i, j) ← +1
23          k ← partition(contextLink(revSuffixLink(l)))
24          x ← revSuffixLink(l)[1]
25          W[x](k, j) ← +1
26  return A, B, V, W[]
```

LEMMA 16.3. The algorithm CONTEXT-PRUNING has a run-time given by

$$\mathcal{O}(n + (|\Sigma| \cdot pdim)^2) \quad \text{where} \quad n = |s|$$

PROOF. Observe that the construction of the suffixtree of s can be done in linear time in the length of s and the reversal of s and the creation of the reverse prefix tree also

takes linear time. This can be achieved by using the Ukkonen algorithm for linear-time suffixtree construction. Further, it holds that

$$CstPartitionCount \cdot CptPartitionCount \ \leq \ (|\Sigma| \cdot pdim)^2$$

which follows from the method of tree pruning used. Hence, the allocation and initialization of A, B, V and $W[x]$ takes time of order $(|\Sigma| \cdot pdim)^2$. The actual counting loop which finally computes A, B, V and $W[x]$ by incrementing entries in the vectors and matrices is done in time linear in the number of leaves of the reverse prefix tree and thus again linear in the input length. □

16.5. Prune-Suffixtree

The following recursive function is the core of the pruning method. It simply traverses a suffix tree in *depth-first* order and checks if the number of leaves hanging off the current node is still higher than a user specified cutoff in which case the node gets assigned a partition number of 0 ("neutrally colored"). If the leaf count drops below that mark or if a leaf is reached, the depth traversal is stopped, the current node and all it's descendants are colored using the current partition number, the partition number is incremented and the current node and partition number are returned. In full detail, the function works as follows:

```
PRUNE-SUFFIXTREE(tree, cut, partition, node)
 1  if #Children(tree, node) > 0
 2     then
 3          if #Leaves(tree, node) > cut
 4             then
 5                  partition(node) ← 0
 6                  R ← {}
 7                  for child ∈ Children(tree, node)
 8                      do
 9                          R ← R ∪ PRUNE-SUFFIXTREE(tree, cut, partition, child)
10                  return R, partition
11             else
12                  for desc ∈ Descs(tree, node) ∪ {node}
13                      do
14                          partition(desc) ← partition
15                  partition ← partition + 1
16                  return {node}, partition
17
18     else
19          partition(node) ← partition
20          partition ← partition + 1
21          return {node}, partition
```

The net result is a partition colored suffix tree. The nodes which are neutrally colored (partition number = 0) do have more than the specified cutoff number of leaves in their subtree. The nodes which have a partition number > 0 assigned have less than the specified cutoff number of leaves hanging off. All nodes having partition number > 0 have all their descendant nodes identically colored.

Usually, the pruning recursion is started from the root node, giving the desired cutoff and a starting partition number 1 as in the following example:

EXAMPLE 16.1.

$$tree \leftarrow \mathsf{cst}(s\$)$$
$$cut \leftarrow 0.1 \cdot \mathsf{Leaves}(tree)$$
$$borderNodes, partitionCount \leftarrow \textsc{Prune-Suffixtree}(tree, cut, 1, root(tree))$$

The function returns the *border node* set, that is the set of nodes which have partition number > 0 but whose parents are neutrally colored. The function also returns the partition count, that is the number of colors that were assigned. Also, the partition count is identical to the number of nodes in the border node set.

LEMMA 16.4. The function PRUNE-SUFFIXTREE has a run-time given by

$$\mathcal{O}(n) \quad \text{where} \quad n = \#\mathsf{Nodes}(tree)$$

PROOF. The function performs a depth-first traversal of the tree which is bounded in run-time by the number of nodes in the tree. The number of nodes in any compact suffixtree of a string of length n is bounded by $2|n|$. □

CHAPTER 17

Screening of 103 estimated OOMs

This section presents experimental results for 103 OOMs estimated using the COPRUMIC learning algorithm described in the previous chapter. Training data was generated using a 3-dimensional OOM (the probability clock). Based on different samples (all of length $1k$) and different pruning cutoffs chosen in the COPRUMIC algorithm, 103 OOMs were estimated and the resulting models were compared to the target OOM. Varying the pruning cutoff in the COPRUMIC algorithm resulted in different indicative and characteristic events of varying complexity. Our goal here was to look out for possible relations between the investigated parameters.

17.1. Investigated Parameters

The figures presented in this section are xy-plots where each point represents a learned model and x and y are taken from a fixed set of parameters that where compared

(1) The *relative entropy rate* was used to objectively measure the quality of the estimated model by measuring the distance in the probability distributions of the estimated and the real model.

(2) The *sample log-likelihood* is the logarithm to the base of 2 of the probability the estimated model gives for the training sample.

(3) The V *matrix condition* gives the condition in the 2-norm of the counting matrix V that is obtained from the sample based on the specific choice of indicative and characteristic events made for estimating the model.

(4) The V *matrix mutual information* gives the mutual information derived from the counting matrix V that is obtained from the sample based on the specific choice of indicative and characteristic events made for estimating the model (see section 6.4).

(5) The *border node count* is a complexity measure of the partitioning (choice of indicative and characteristic events). It is given as the number of colored nodes that have neutrally colored parent nodes in the colored context graph for the sample and the partitioning chosen (compare definition 14.2).

17.2. Discussion of Results

This section gives a very short discussion of the results represented. Perhaps the most important result is illustrated in figure 17.1. Here, the relative entropy rate of the estimated models with respect to the target model is plotted versus the log-likelihood given to the training sample by the estimated models. The thick line drawn (by hand) indicates a mean value of the dots printed. The dashed vertical line indicates the location where the mean relative entropy rate reached a minimum (sample likelihood $p = 2^{-975}$). Obviously, this is not where the log-likelihood was maximal (sample likelihood $p = 2^{-955}$)

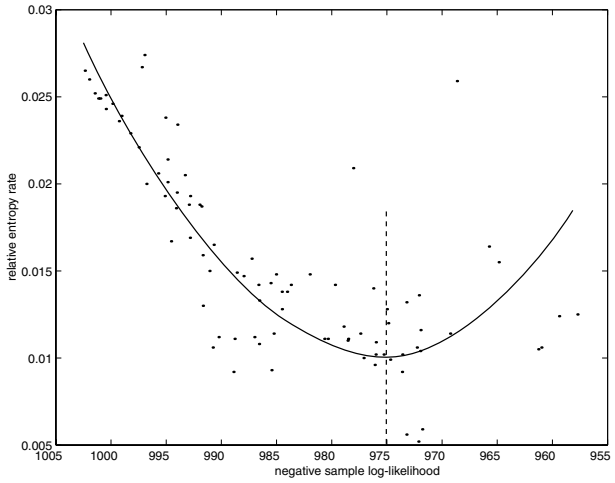

FIGURE 17.1. Sample log-likelihood versus relative entropy rate.

and thus the sample would have had highest probability. One may interpret this minimum as the balance point between model precision achieved by increasing the log-likelihood and overfitting the sample.

Other aspects that may be seen from the figures are:

(1) Neither the V matrix condition nor the V matrix mutual information alone can indicate a low relative entropy rate, that is good model quality (see figures 17.2 and 17.3) or a high sample log-likelihood (see figures 17.5 and 17.6). In other words, model precision as indicated by a low V matrix condition or a high V matrix mutual information alone is not enough for high model quality, since overfitting may happen impairing the model's generalization abilities.

(2) Border node count alone, as a measure of model complexity is also not sufficient in indicating a high model quality (see figure 17.4) or a high sample log-likelihood (see figure 17.7).

(3) Relative entropy rate and Hellinger rate give qualitatively similar results (see figure 17.8).

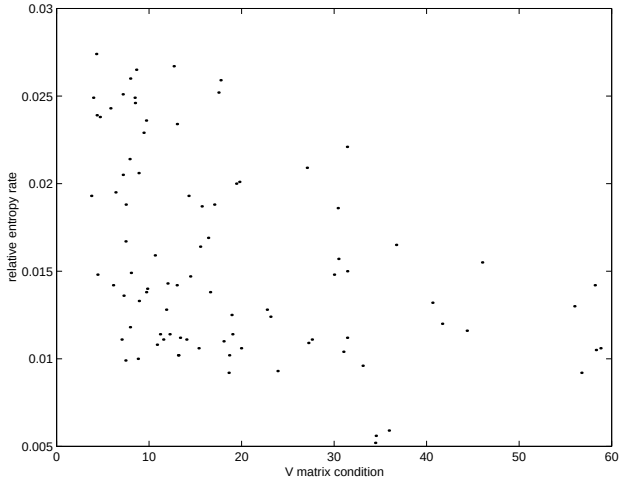

FIGURE 17.2. V matrix condition versus relative entropy rate.

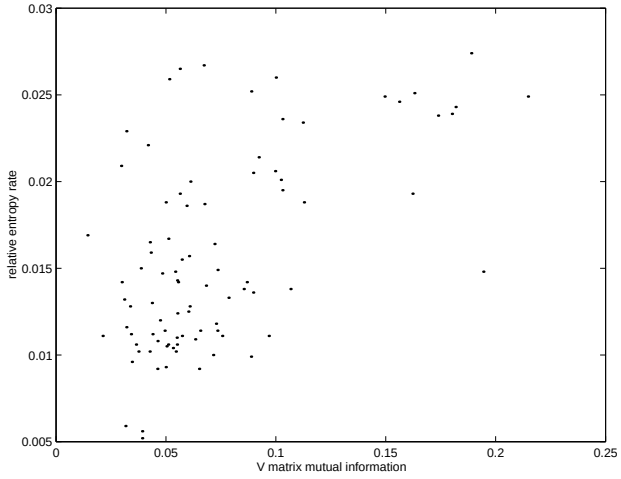

FIGURE 17.3. V matrix mutual information versus relative entropy rate.

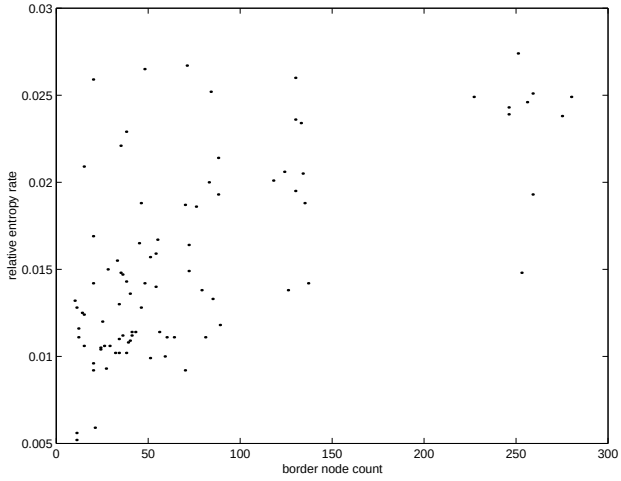

FIGURE 17.4. Border node count versus relative entropy rate.

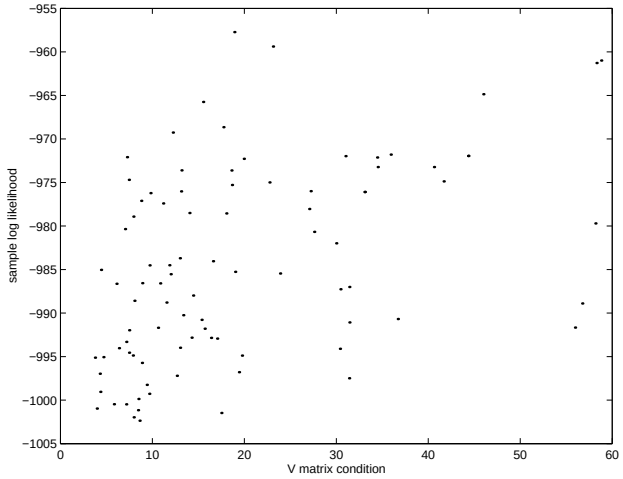

FIGURE 17.5. V matrix condition versus sample log-likelihood.

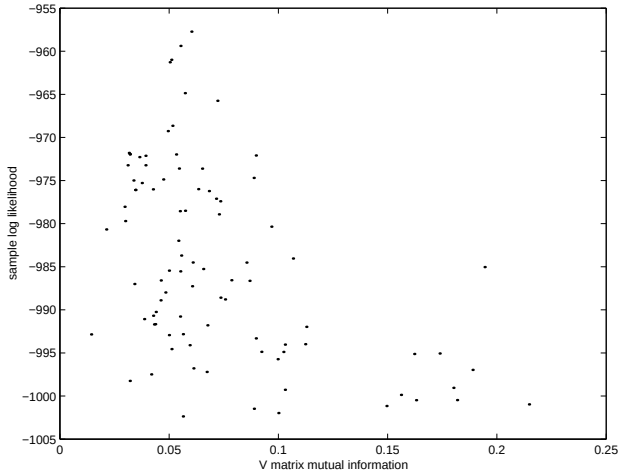

FIGURE 17.6. V matrix mutual information versus sample log-likelihood.

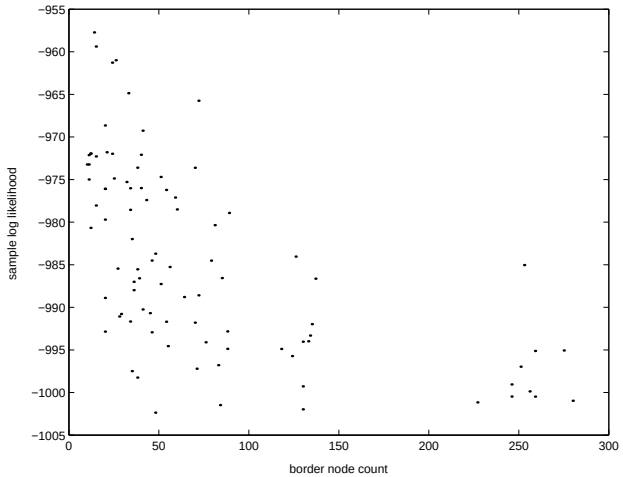

FIGURE 17.7. Border node count versus sample log-likelihood.

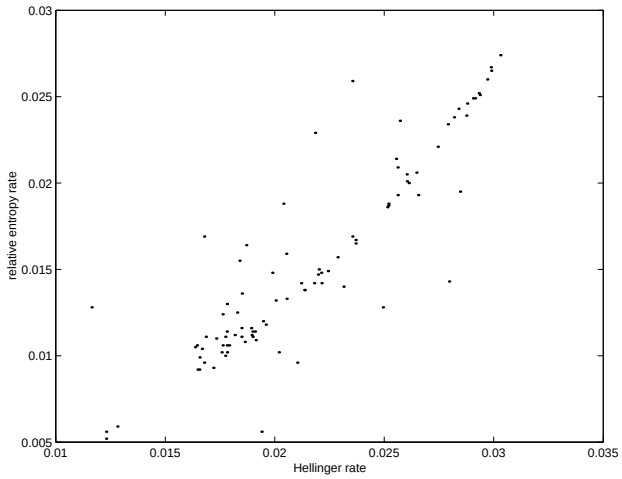

FIGURE 17.8. Hellinger rate versus relative entropy rate.

Part 4

Observable Operater Modeling Kit

A major part of the work done in this thesis consisted of developing a software package for the simulation, analysis and training of OOMs.

The software developed, the *Observable Operator Modeling Kit* (OMK) was written from scratch and consists of 30.000 lines of portable C++ code. The package is well documented and already used internally in the lab. For example an OOM estimation method [**Kre**] has been developed using the facilities provided by OMK and based on the *context pruning* method.

In the sections of this part I will first outline the functionality and overall architecture of OMK. Then I describe the different modules in higher detail with respect to function, design and implementation. I close with a short outlook on future work.

CHAPTER 18

Functionality, Architecture and Design Paradigm

18.1. Functionality

The OMK provides a number of orthogonal facilities that together lay the foundation for the simulation, analysis and training of OOMs. In particular, the kit provides functionality for

(1) simulating OOMs
(2) information theoretic analysis of stochastic processes
(3) numerical linear algebra and non-linear optimization
(4) suffix tree and context graph construction and analysis
(5) OOM estimation method based on mutual information

The OMK is targeted towards OOM applications. However, the software has a *modular* design which allows to reuse some of the facilities of the OMK in applications unrelated to OOMs. In particular, the suffix tree, context graph and numerical modules are self-contained and could be useful elsewhere.

18.2. Architecture

The static structure of the OMK is documented and specified in figures 19.1, 19.2, 19.3, 19.4, 19.5 using the class diagram notation of the Unified Modeling Language (UML) [**Oes01**]. The UML is a semi-formal graphical notation for specifiying object-oriented systems. It is widely accepted in the field of object-oriented technology and standardized by the Object Management Group (OMG) [**Gro**].

18.3. Design Paradigm

The OMK was designed based on the paradigm of *generic programming*. Generic programming is both a design paradigm and a implementation method. It is a modern concept younger than object-orientation ([**Ale01**]).

> "My working definition of generic programming is 'programming with concepts', where a concept is defined as a family of abstractions that are all related by a common set of requirements. A large part of the activity of generic programming, particularly in the design of generic software components, consists of concept development - identifying sets of requirements that are general enough to be met by a large family of abstractions but still restrictive enough that programs can be written that work efficiently with all members of the family." (from [**Mus**])

The design aspect of generic programming stresses the importance of *orthogonality*. This means factoring domain aspects such that no domain abstraction overlaps arise and all domain abstractions may be freely combined. We will quickly see an example.

The implementation aspect of generic programming is realized by the programming language in use. In C++ ([**Str00**]), the *template* language facility is particulary effective for implementing software in a generic programming style. The wonderful thing in doing so is that it combines powerful abstraction without sacrificing efficiency. This is different

from classical object-oriented programming, where much use is made from run-time polymorphism and virtual functions which can compromise performance severely. It is fun to note that in Java, where no language facilities for generic programming are available, a recent effort is to introduce so-called Java-generics trying to retrofit crippled support for generic programming. It is important to note, that C++ is far from providing simple and complete generic programming facilities. C++ is a monster. Clean support from the language core is available in functional programming language based on typed Lambda calculus like e.g. ML or Haskell.

Probably the best known example of generic programming in the C++ community is the widely deployed *C++ Standard Template Library (STL)* ([**PSLM01**]) which is part of the C++ language standard. A considerable part of the library is built on only three concepts

- **container**
- **iterator**
- **algorithm**

Iterators are the glue between algorithms and containers. That is, algorithms working on containers are insulated by the use of iterators.

How is generic programming deployed in OMK? Like the STL has containers, iterators and algorithms, the OMK includes the following concepts

- **process**
- **sample**
- **measure**

We will not go into the details of these concepts right now (see subsection 19.1.1) but just give an impression how generic programming looks like with OMK. The following code snippets shows a signature of a typcial function template in OMK:

```
template<class P, class Q, class F, class S>
     double pfunctional (P& p_process,
                         Q& q_process,
                         const F& measure,
                         S& sample);
```

Now, for example, we could use the function *pfunctional* to compute the relative entropy (= measure) between two OOMs (= process) on the basis of a sample generated from one of the processes itself (= sample). This illustrates how orthogonal domain abstractions (process, sample and measure) are combined a flexible and transparent way. It is important to note, that this is achieved without losing efficiency. I will not go into the details of C++ compiler technology and template instantiations here, which would be necessary to provide sufficient arguments to support the efficiency claim. The reader may consult [**Str00**] or the compiler vendor documentation for details on this topic.

CHAPTER 19

Modules

19.1. OOM Simulation and Analysis

Given a particular OOM, the OMK provides for

(1) generate samples according to the distribution defined by the OOM
(2) evaluate probabilities for the OOM emitting given strings

The generation and evaluation functions together will be called *OOM run-mode*. The implementation of OOM run-mode in the OMK has a number of special features:

- efficient support for large alphabets
- efficient support for high dimensional OOMs
- numerical precision and alphabet type are user defined
- factor 200 speed-up over previous Matlab implementation
- can use vendor optimized numerical libraries (e.g. [**int**]) for LAPACK/BLAS)
- uses state of the art pseudo-random number generator ([**MN98**])

In the course of this thesis I found the efficient and reliable simulation of OOMs indispensable for any successful work directed towards experimental analysis and training of OOMs. The OMK provides this basis.

19.1.1. Design and Implementation. In OMK there are three basic domain abstractions related to the simulation and analysis of stochastic processes:

(1) process
(2) sample
(3) measure

An OOM is a specific *process*. Other available processes include the *iid*-process and the *surrogate*-process. This is illustrated in figure 19.3 using the UML class diagram notation. The fact that the three classes `Oom`, `SurrogateProcess` and `IudProcess` are all specific realizations of the abstract *process* domain concept is expressed via the inheritance relation to the interface `IProcess`. However, the interface base class `IProcess` only serves documentation purposes and is deliberately *not* implemented since the OMK was designed using generic programming and compile-time polymorphism in mind as explained in section 18.3.

Similarily to the *process* concept, the domain concept *sample* is implemented by different classes providing different realizations of the domain concept (figure 19.4). The class `IudSample` provides samples generated independently and identical distributed and the class `TotalSample` provides for all strings of fixed length from the given alphabet. The samples generated from class `ProcessSample` are produced by a stochastic process that the user provides. In terms of generic programming, the class `ProcessSample` is parametrized by a process type. This allows for the generation of samples based on OOMs for example.

The power of generic programming becomes visible when we look at the third domain concept: *measure*. A measure is a functional of the probability distribution of a stochastic process. The measure is evaluated on the basis of a sample. The relations may become clear from figure 19.5.

An example would be: evalutate the relative entropy between two OOMs on the basis of a fixed number of iid samples. We can see how the domain concept of measure combines entities (OOM, iid sample) of the orthogonal doman concepts process and sample to compute some value (relative entropy).

Generic programming in this situation not only leads to a clear design by separating orthogonal aspects, but when implemented using the C++ facility of *templates* this also results in highly efficient code. This is very different from a OO style design, where we could have also separated aspects, but the resulting implementation would have been crippled by performance problems due to the use of run-time polymorphism and *virtual functions*.

19.2. Numerics

The C++ language was choosen as the main implementation language because of its unique combination of high performance (if done right) and modern abstraction facilities (classes and templates). In particular, C++ shines at *data structures* and *applied discrete mathematics*.

However, in addition OMK also needed basic and some advanced numerical functions as well:

- matrix-matrix product, matrix-vector product, dot product
- matrix inversion, solving systems of linear equations
- matrix norms and condition estimation
- singular value decomposition (SVD)
- non-linear constrained optimization

Implementing stable and efficient numerical codes can be tricky and time consuming. Instead, I decided to build on the experience of the Fortran community with numerical codes by developing an C++ interface layer on top of standard Fortran numerical libraries.

The Fortran community has 40+ years of numerical codes know-how and a number of defacto standard libraries are in wide use:

(1) BLAS
(2) LAPACK
(3) PORT

All three libraries have a Fortran interface and are implemented in Fortran77.

19.2.1. BLAS/LAPACK. BLAS stands for *Basic Linear Algebra Subprograms*, which also describes the functionality provided by the library. LAPACK, which stands for *Linear Algebra PACKage* is a successor to LINPACK and EISPACK, all of which are concerned with numerical linear algebra beyond simple matrix and vector arithmetics:

> "LAPACK is written in Fortran77 and provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems." [**net**]

BLAS and LAPACK are available on many platforms both in open-source implementations ([**net**]) and in vendor optimized versions. For example, Intel offers a version aggressively optimized for Intel processors, the *Intel Math Kernel Library* ([**int**]):

> "The Intel Math Kernel Library (Intel MKL) is composed of highly optimized mathematical functions for math, engineering, scientific and financial applications requiring high performance on Intel platforms.

> Intel MKL contains LAPACK, the basic linear algebra subprograms (BLAS), and the extended BLAS (sparse)."

Also, many commercial math applications like Matlab and Mathematica rely on LAPACK/BLAS and today major Linux distributions already contain precompiled, easy to install versions of LAPACK/BLAS. All in all, BLAS/LAPACK was chosen as OMK's underlying numerical library because

(1) it is widely known and professionally deployed
(2) has stable, efficient and proven implementations available
(3) provides both basic and advanced numerical linear algebra functionality

Looking back, this decision proved to be right.

19.2.2. PORT. Another particular OOM learning method developed in the lab ([**Kre**]) required non-linear constrained optimization functionality besides the functionality already implemented in the OMK. To enable these efforts, I further extended the numerics module by implementing an interface layer on top of another advanced Fortran library.

PORT is a Fortran77 numerical package that includes functionality for non-linear optimization. PORT is made available by *Bell Labs* ([**por**]) and is free to use for non-commerical purposes:

> "The PORT Mathematical Subroutine Library (third edition) is a collection of Fortran 77 routines that address many traditional areas of mathematical software, including approximation, ordinary and partial differential equations, linear algebra and eigensystems, optimization, quadrature, root finding, special functions, and Fourier transforms, but excluding statistical calculations. PORT stands for Portable, Outstanding, Reliable, and Tested."

19.2.3. Design and Implementation. To interface Fortran libraries to C++ code and to provide numerical functions the OMK contains a numerics module. The OMK numerics module consists of

(1) vector and matrix classes
(2) LAPACK/BLAS routine wrappers
(3) PORT routine wrappers
(4) additional routines

The vector and matrix classes are fairly light-weight, templated C++ classes parametrized over their numeric type

```
template<class T> Matrix { ... };
template<class T> Vector { ... };
```

The matrix and vector classes handle the Fortran/C++ issues of different element numbering and different storage layout. Internally, the classes use Fortran-style *column major order* memory layout for storing arrays. Thus, no efficiency is lost because no column to row major order conversion must be performed.

Based on the matrix and vector abstractions, the interface to the various numerical codes has a signature like in the following example

```
template<class T> inline Vector<T>& svd (Matrix<T>& a, Vector<T>& s);
```

The wrapper code internally handles the Fortran/C++ issues of different calling conventions. The vector and matrix classes together with the routine wrappers serve as an interface layer to the Fortran libraries. The layer was designed to be easy to use and

extend and preserve the efficiency of Fortran libraries.

The interface to the PORT library is very similar and likewise builds on the vector and matrix classes. An example of a non-linear optimization routine:

```
/**
 * Abstract driver for non-linear minimization subject to simple
 * constraints (component bounds) where no derivates are needed.
 * Note: this is a simplified version, where scaling is internally
 * computed as x0*_i = 1. / x0_i.
 *
 * @param f          function object for objective function
 * @param x0         initial guess or start vector
 * @param bounds     a 2 x dim(x0) matrix containing lower and upper bounds
 * @param info       parameter and knob settings for optimization; return info
 * @return           f(x*) at computed minimum x*
 */
template<class T, class F>
inline
T
mnfb (F& f,
      Vector<T>& x0,
      const Matrix<T>& bounds,
      OptInfoSet<T>& info);
```

Besides the wrapper routines, I developed a number of additional routines for purposes of

(1) linear regression
(2) matrix reductions based on
    (a) euclidean distance
    (b) mutual information
    (c) matrix condition

where (2.b) is used in the implementation of a simple OOM learning method ("context pruning + greedycluster").

A UML class diagram illustrating the static class structure behind the OMK numerics module is given in figure 19.2. In summary, the gap between a modern object-oriented generic language (C++) and reliable and efficient numerical codes (Fortran) was successfully bridged. Also, as the infrastructure is in place, future extensions like wrapping up more routines or more Fortran libraries are easy to do.

19.3. Suffix Trees and Context Graphs

Suffix trees are terrific data structures with broad applications. For the purposes of OOM learning, I extended the notion of suffix trees in two aspects:

(1) *colorings* of suffix trees
(2) reverse-prefix tree / suffix tree pairs plus context links (= *context graphs*)

(1) is necessary to represent both the input training sample and the chosen partitioning within the same data structure. (2) is important to represent both partitionings of past and future context giving rise to indicative and characteristic events within the same data structure. All this was already discussed in the part about suffix trees and context graphs.

For OMK, I implemented suffix trees based on the linear-time, linear-space Ukkonen algorithm ([**Ukk95**]) and context graphs with above characteristics.

The implementation has a couple of unique features:

- string type is user defined
- efficient support for large alphabets up to 10.000's of symbols
- extensive built-in tree statistics

In particular, the efficient support for large alphabets enables a number of interesting experiments. For example, I have built and analyzed suffix trees and context graphs over *word alphabets* where the symbols are taken as the words appearing in a natural language text.

A suffix tree can be built from a string and tree statistics computed and printed as simple as this

```
std::string input;
...
UStree<std::string> stree (input.data (), input.length ());
UStreeFull<std::string> streeFull (stree);
UStreeFull<std::string>::UStreeStatistics stats (streeFull.getStats ());
stats.print (std::cout);
```

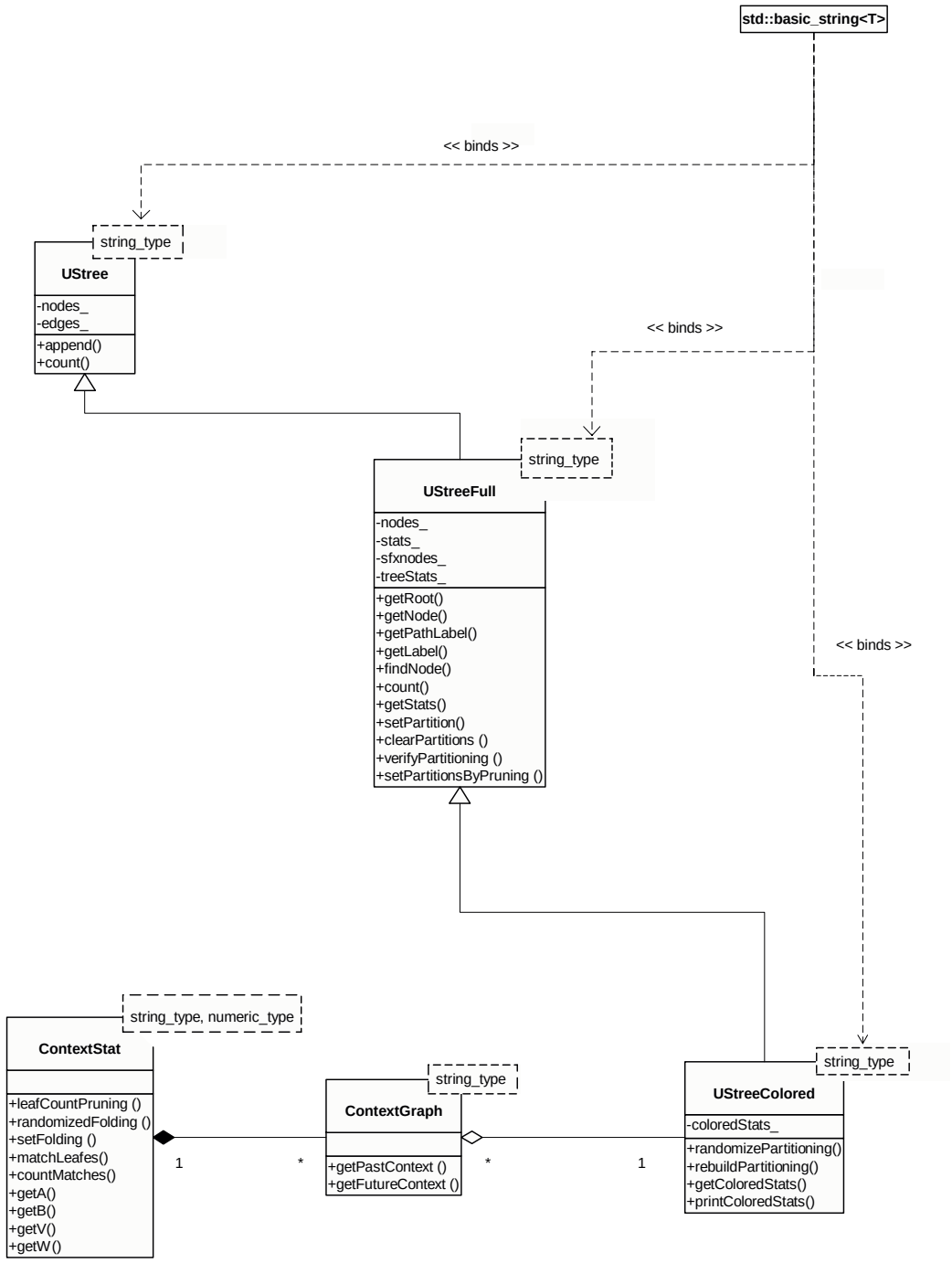

FIGURE 19.1. OMK class diagram - suffixtree and context module

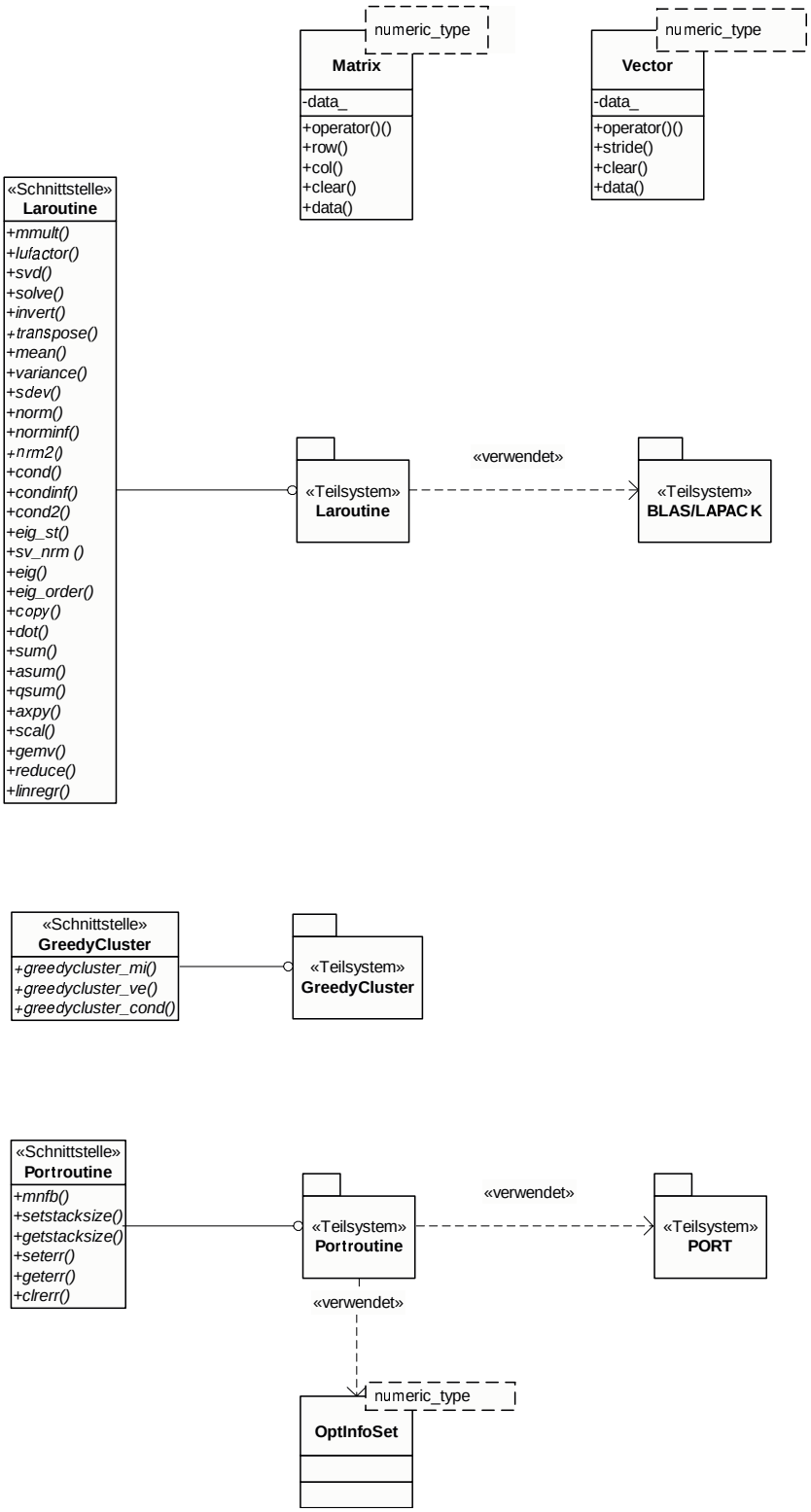

FIGURE 19.2. OMK class diagram - numerics module

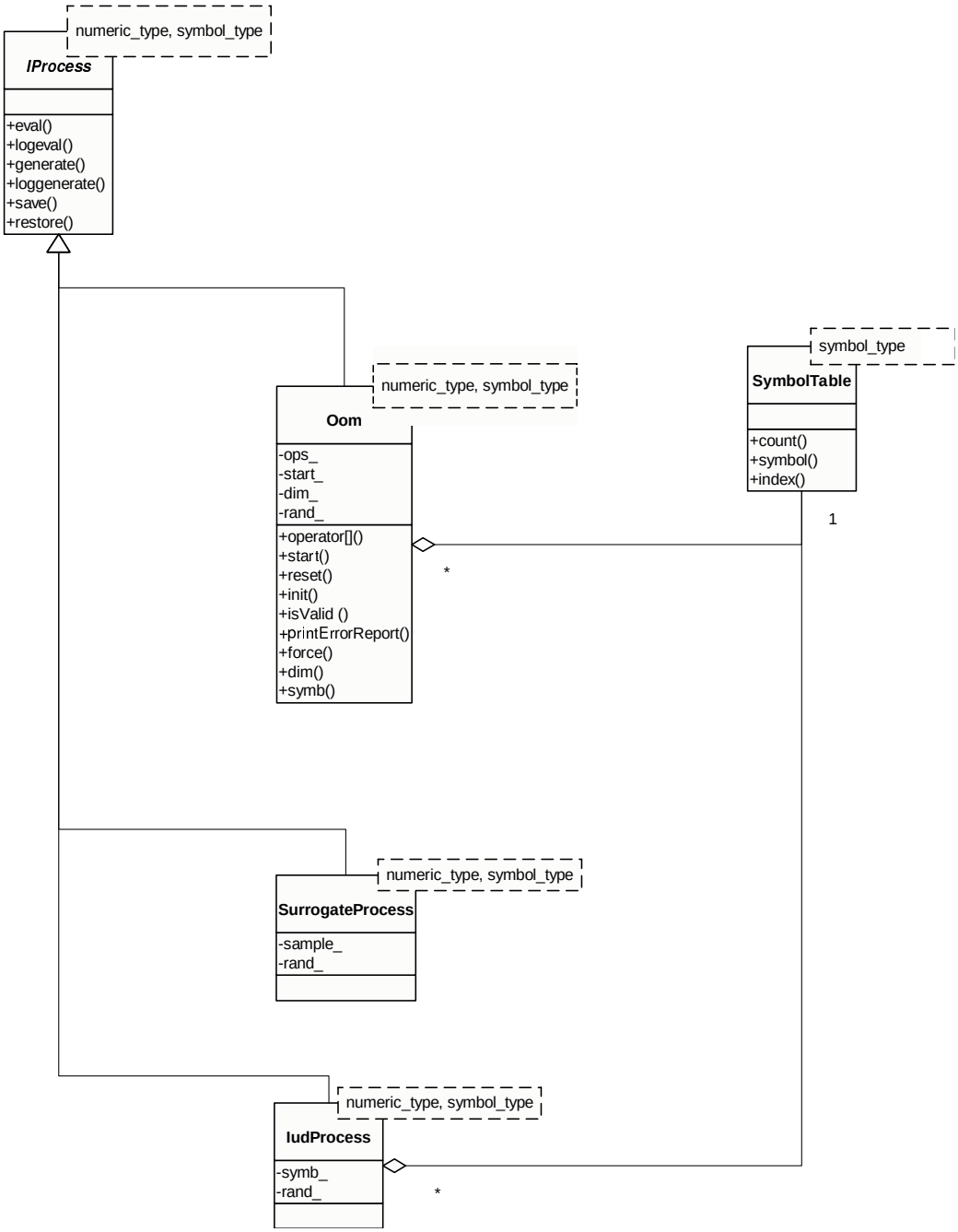

FIGURE 19.3. OMK class diagram - OOM and process module

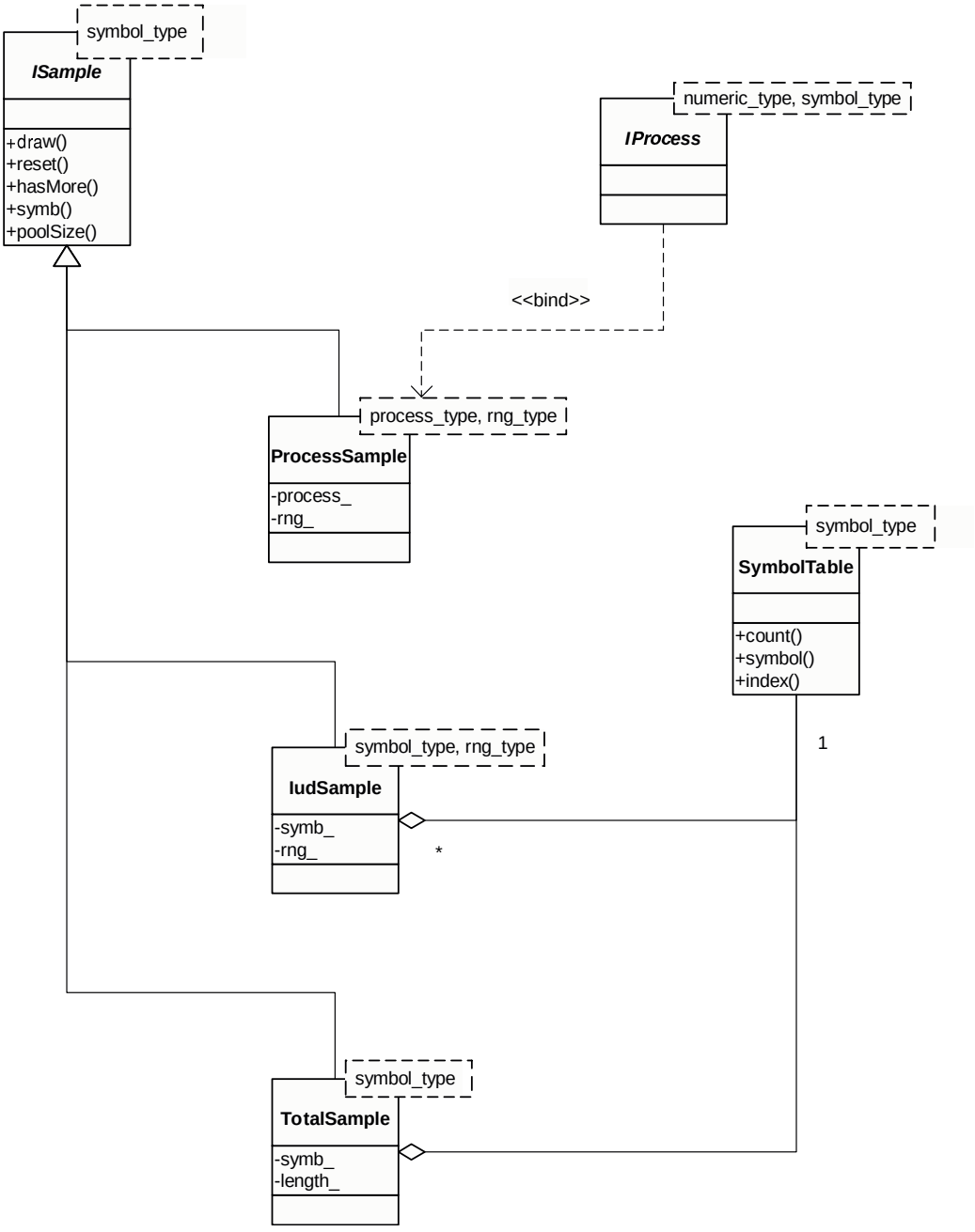

FIGURE 19.4. OMK class diagram - sample module

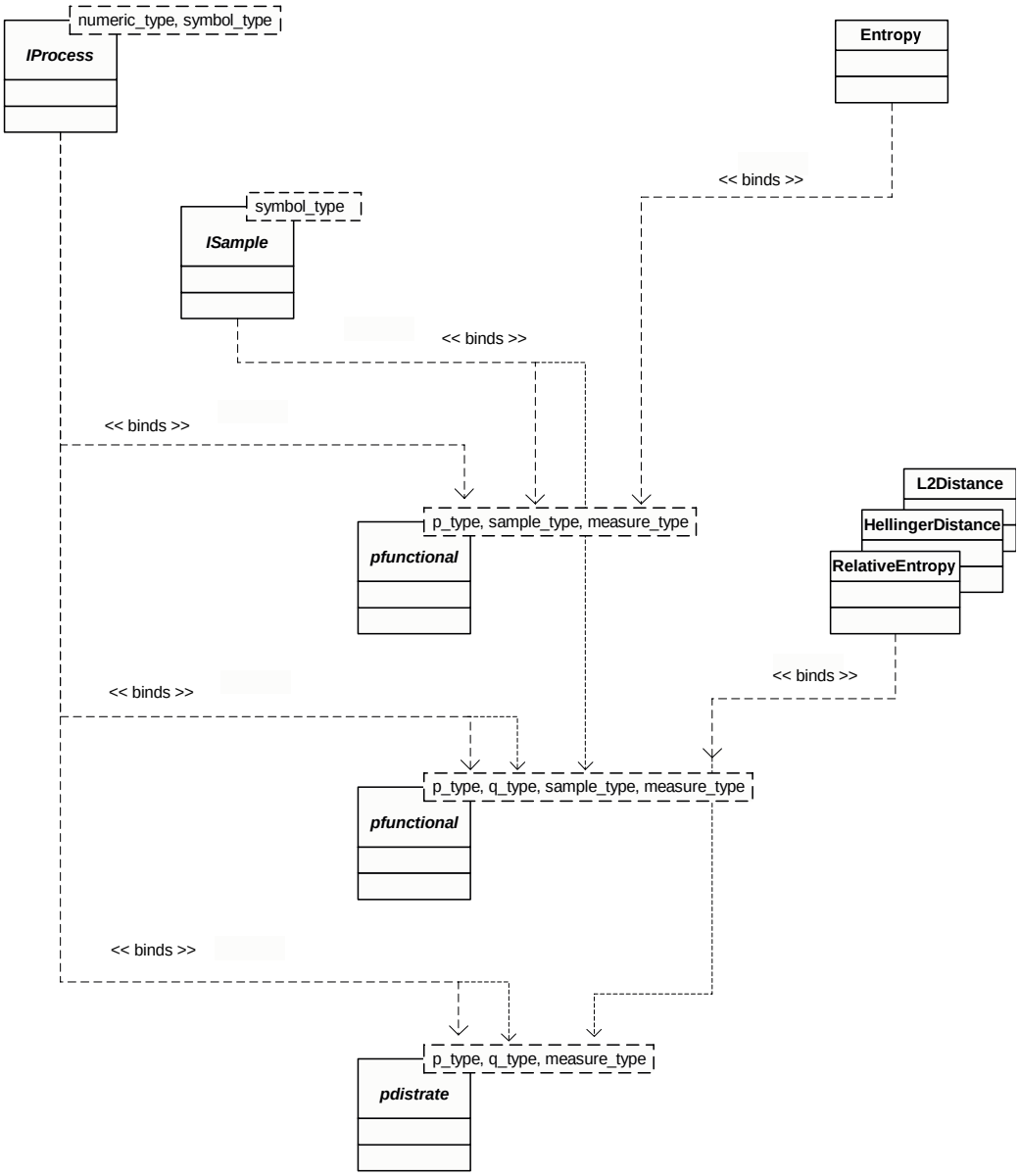

FIGURE 19.5. OMK class diagram - pfunctional module

CHAPTER 20

Sftree

Presently, OMK is largely a library of classes for developing OOM applications and adresses software developers and researchers. In the mid-term, a number of end-user tools will be created that can be deployed without programming. A first tool developed in this direction is **sftree**.

sftree is a command line tool that uses the suffix tree facilities of OMK to create a compact suffixtree or reverse prefix tree from given input, compute detailed tree statistics and output tree structure and statistics in different formats.

I will shortly introduce the provided functionality along with the command line options and explain the program's output. After that, I discuss some results from applying the tool to time-series data of various types.

20.1. Sftree Usage

The tool was developed during the initial phase of this thesis to explore the capabilities and characteristics of suffix trees built from sample data of various kinds. It is quite flexible and also capable of computing a rich set of tree statistics. In particular **sftree** can do:

- read input from file or from **stdin**
- compute suffixtree or reverse prefix tree of input
- output tree structure in text format, csv (comma separated value) or in pstree [1]
- compute histograms over symbol alphabet and edge labels
- compute tree level expansion ratios
- compute summary statistics over various tree aspects

The tool is efficient - it is no problem to analyze inputs of length 1M - 10M. Help is available by invoking the tool without any parameters, which gives complete list of command line options

```
toberste@stalker% sftree -h

sftree v1.4 - Copyright 2002 Tobias Oberstein.
<tobias.oberstein@ais.fraunhofer.de>

sftree [-servhOL] [-p | -P] [-lLabelLength] [-mMode] (-iInputFile | -IInput)
[-oOutputFile] [-tSeparatorChar]

   -s              print statistics
   -e              print edge label histogram
   -r              print leaf count histogram
   -v              validate suffix tree structures
   -h              print this help message
   -O              suppress output
   -L              print label in edge compressed format
```

[1] pstree is a latex package for typesetting trees; I used sftree and pstree to produce the figures 13.5 and 13.5

```
-p               create the reverse prefix tree instead
-P               like -p, but do not include the last  character (sentinel)
                 in reversal of input
-lLabelLength    print edge labels up to given length
-yScale          scaling factor for pstree mode (\scalebox{Scale})
-mMode           output mode: either plain, csv or pstree
-iInputFile      input file to read from
-IInput          read input from arg, like -I"abba*"
-oOutputFile     output file to write to
-tSeparatorChar  separator character in csv mode
```

A typical invocation might look like this

```
sftree -Os -i datafile
```

which is also what I used in generating the output for various time-series data examples in the next section.

20.2. Sftree Statistics Output

20.2.1. Alphabet Histograms. sftree computes histograms over the symbol alphabet occurring in the input. The alphabet histogram just gives the number of occurrences of the different symbols in the alphabet of the input. The alphabet of the input is just all symbols occurring anywhere in the input. Currently, the input is interpreted as one byte equals one symbol. Thus, it is not possible to read the input having two bytes interpreted as one symbol each, what would be possible from the underyling OMK suffix tree classes though.

20.2.2. Edge Label Histogram. Labels are finite strings attached to tree edges. All labels are substrings of the original input. The edge histogram gives us the number of occurrences of labels attached to tree edges. In general, a given edge label will occur more than once within the tree.

20.2.3. Tree Level Expansion Ratios. sftree also computes so-called tree level expansion ratios. This works as follows. Suppose the symbol alphabet of the input would be Σ. Then we can imagine a fully expanded, maximal tree where all nodes have $|\Sigma|$ children and the edges to those children are labeled with one symbol each, for all the different symbols in Σ. Observe, that the suffixtree constructed for the input will be a subtree of the full tree then, if we expand compacted edges again. Now we may ask what fraction of nodes in the full tree residing at some given level (node depth from the root) is also occupied by the suffixtree subtree for the input. This fraction is the tree level expansion ratio and is obviously a number between 1 and 0.

20.2.4. Summary Statistics. sftree computes so-called summary statistics, that is

- mean
- standard deviation
- minimum
- maximum

over three node sets

- all internal nodes without the root node
- all node without the root node
- all leaf nodes

and for different node data

- descendant, child and leaf count
- path, node and fork depth

For a fixed node k, the descendant, child and leaf count give the number of nodes in the subtree below k, the number of children of k and the number of leaves residing somewhere in the subtree below k. The statistics for counts are only computed for the full node set.

The path and node depth is the cumulative length of all concatenated labels from the root down to node k and the number of nodes from the root down to node k. The fork depth is the path depth of the parent of node k plus 1. The fork depth is computed only for leaf nodes, whereas the path and node depth are computed for all nodes and all nodes / leaves only.

The idea behind the fork depth is the following. What is the number of symbols we must observe, until we know already the complete context? Obviously, if in the suffixtree we have arrived at a parent node of some leaf, then observing one more symbol will disambiguate the edge we have to travel to the final leaf. Thus, we have taken the path length of the parent node of the leaf plus 1 symbols to fully determine the context. This exactly is teh fork depth. For example, a mean fork depth of 10 means that on average we must be presented 10 consecutive symbols of the input such that we already know the exact position of the presented string within the input.

20.3. Analyzing Time-Series Data with Sftree

This section presents output generated from sftree for various samples of time-series, all of length 10k. The motivation was to gain experience with suffixtrees and insight into how structure and statistics of suffixtrees are influenced by the characteristics of the input strings. A diverse set of time-series was used to build suffixtrees from and compute tree statistics. The results presented in this section are for the following time-series:

- Independent Identically Distributed
- Short-Long Memory HMM
- Three dimensional OOM
- The Probability Clock
- Discretized Mackey-Glass
- Discretized Linear Congruential
- The "showcase" 2-dimensional HMM
- Independently Distributed

The results of suffixtree summary statistics for leaf node depth and leaf fork depth is given in figures 20.1 and 20.2 respectively.

20.3.1. Independent Identically Distributed. The sample was generated by drawing symbols from the alphabet $\Sigma = \{a, b, c\}$ independently and with equal probability.

```
Standard Statistics
-------------------------------------------------
input length       10001
alphabet size      4
alphabet           abc~
nodes              17346
leafs              10001
internal nodes     7345

Alphabet Histogram
-------------------------------------------------
a                   3370
b                   3312
c                   3318
```

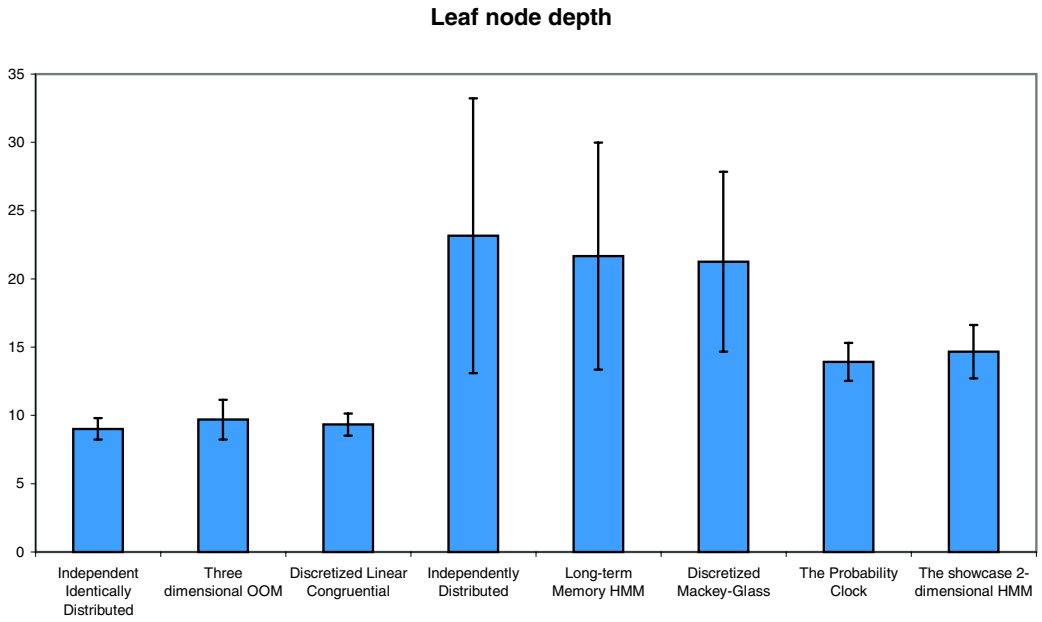

FIGURE 20.1. Leaf node depth mean and standard deviation

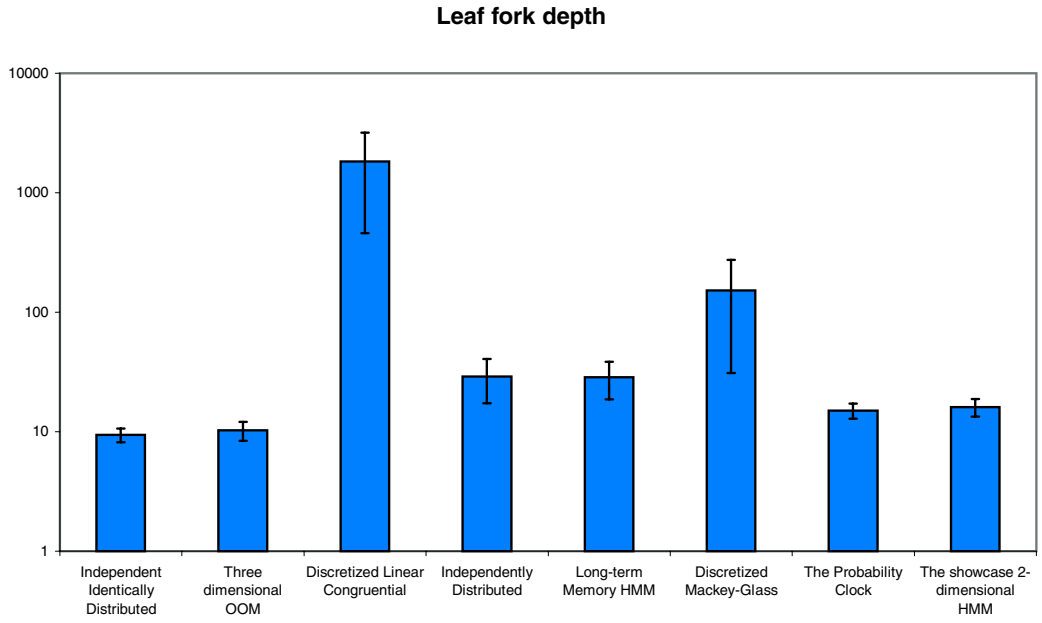

FIGURE 20.2. Leaf fork depth mean and standard deviation

```
~                    1

Summary Statistics of Suffix Tree
-------------------------------------------------
                   mean     stddev       min       max
desc count *      17.47     141.95         2      5864
leaf count *      10.92      81.84         2      3370
child count*       2.36       0.48         2         4
path depth +    2886.84    3300.50         1     10001
```

```
node depth +     8.40      1.21        1        12
node depth #     9.02      0.79        1        12
fork depth #     9.39      1.22        1        16

* : stats over internal nodes (not the root)
+ : stats over all nodes (not the root)
# : stats over leaf nodes

Tree Level Expansion
--------------------------------------------------
level expansion ratio

1 1.000000
2 0.625000
3 0.437500
4 0.320312
5 0.238281
6 0.178223
7 0.132324
8 0.078674
9 0.030144
10 0.008792
11 0.002316
12 0.000590
13 0.000148
14 0.000037
15 0.000009
16 0.000002
```

20.3.2. Short-Long Memory HMM. The sample was generated by an OOM that was obtained by transforming the HMM as defined in equations 20.1, 20.2 for $\epsilon = 0.1$ into an OOM as indicated in 20.3. The OOM was allowed a 10k warm-up to wash out any numerical non-stationarity.

$$M = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 1-\epsilon & \epsilon & 0 \\ 0 & 0 & 0 & 1 \\ \frac{\epsilon}{2} & 0 & \frac{\epsilon}{2} & 1-\epsilon \end{pmatrix} \tag{20.1}$$

$$O_a = \begin{pmatrix} 1 & & & \\ & 0 & & \\ & & 0 & \\ & & & 0 \end{pmatrix}, \quad O_b = \begin{pmatrix} 0 & & & \\ & 0 & & \\ & & 1 & \\ & & & 0 \end{pmatrix}, \quad O_c = \begin{pmatrix} 0 & & & \\ & 1 & & \\ & & 0 & \\ & & & 1 \end{pmatrix} \tag{20.2}$$

$$\tau_x = M^T \cdot O_x, \ x \in \{a, b, c\} \quad \text{and choose } w_0 \text{ such that} \quad M^T \cdot w_0 = w_0 \tag{20.3}$$

```
Standard Statistics
--------------------------------------------------
input length       10001
alphabet size      4
```

```
alphabet           abc~
nodes              18855
leafs              10001
internal nodes     8854

Alphabet Histogram
--------------------------------------------------
a                   339
b                   651
c                   9010
~                   1

Summary Statistics of Suffix Tree
--------------------------------------------------
                  mean    stddev      min       max
desc count *     42.26    394.12        2     16988
leaf count *     23.35    209.28        2      9010
child count*      2.13      0.35        2         4
path depth +   2664.65   3253.95        1     10001
node depth +     20.85      8.33        1        61
node depth #     21.67      8.32        1        61
fork depth #     28.54      9.91        1        68

* : stats over internal nodes (not the root)
+ : stats over all nodes (not the root)
# : stats over leaf nodes

Tree Level Expansion
--------------------------------------------------
level expansion ratio

1 1.000000
2 0.375000
3 0.171875
4 0.074219
5 0.034180
6 0.015137
7 0.006226
8 0.002472
9 0.000954
10 0.000343
11 0.000118
12 0.000039
13 0.000013
14 0.000004
15 0.000001
16 0.000000
.. ..
68 0.000000
```

20.3.3. Three dimensional OOM. Data was generated by a 3-dimensional, synthetically constructed OOM on the alphabet $\Sigma = \{a, b, c\}$ as defined in equations 7.3 and **??**. This is also one of the reference OOMs used in the examples of P-distance measures. The OOM was allowed a 10k warm-up to wash out any numerical non-stationarity.

```
Standard Statistics
-------------------------------------------------
input length       10001
alphabet size      4
alphabet           abc~
nodes              17627
leafs              10001
internal nodes     7626

Alphabet Histogram
-------------------------------------------------
a                   3812
b                   2116
c                   4072
~                   1

Summary Statistics of Suffix Tree
-------------------------------------------------
                  mean    stddev      min       max
desc count *     18.60    152.27        2      7199
leaf count *     11.40     86.32        2      4072
child count*      2.31      0.46        2         4
path depth +   2841.28   3293.59        1     10001
node depth +      9.05      1.73        1        16
node depth #      9.69      1.46        1        16
fork depth #     10.24      1.88        1        19

* : stats over internal nodes (not the root)
+ : stats over all nodes (not the root)
# : stats over leaf nodes

Tree Level Expansion
-------------------------------------------------
level expansion ratio

1 1.000000
2 0.625000
3 0.437500
4 0.320312
5 0.238281
6 0.176758
7 0.116760
8 0.057755
9 0.022072
10 0.007141
11 0.002064
12 0.000558
13 0.000145
14 0.000037
15 0.000009
16 0.000002
17 0.000001
.. ..
19 0.000000
```

20.3.4. The Probability Clock. The classic: probability clock.

```
Standard Statistics
-------------------------------------------------
input length       10001
alphabet size      3
alphabet           ab~
nodes              19988
leafs              10001
internal nodes     9987

Alphabet Histogram
-------------------------------------------------
a                   4809
b                   5191
~                   1

Summary Statistics of Suffix Tree
-------------------------------------------------
                 mean    stddev      min       max
desc count *    23.88    201.78        2     10375
leaf count *    12.95    100.95        2      5191
child count*     2.00      0.03        2         3
path depth +  2508.65   3223.71        1     10001
node depth +    12.93      1.97        1        19
node depth #    13.93      1.39        1        19
fork depth #    15.00      2.15        1        26

* : stats over internal nodes (not the root)
+ : stats over all nodes (not the root)
# : stats over leaf nodes

Tree Level Expansion
-------------------------------------------------
level expansion ratio

1 1.000000
2 0.555556
3 0.333333
4 0.209877
5 0.135802
6 0.089163
7 0.058985
8 0.039018
9 0.025860
10 0.016952
11 0.010680
12 0.006164
13 0.003132
14 0.001397
15 0.000556
16 0.000206
17 0.000073
18 0.000025
19 0.000008
```

```
20 0.000003
21 0.000001
.. ..
26 0.000000
```

20.3.5. Discretized Mackey-Glass. Data generated from discretized Mackey-Glass time-series. The Mackey-Glass time series itself was computed using a 4th-order Runge-Kutta method (Roger Jang, EECS Dept., UC Berkeley, 1992). The resulting real valued time series was discretized into the alphabet $\Sigma = \{a, b, c\}$ using the threshold levels 'a' if > 1.0, 'b' if > 0.8 and 'c' otherwise.

```
Standard Statistics
-------------------------------------------------
input length       10001
alphabet size      4
alphabet           abc~
nodes              19964
leafs              10001
internal nodes     9963

Alphabet Histogram
-------------------------------------------------
a                   4539
b                   2385
c                   3076
~                   1

Summary Statistics of Suffix Tree
-------------------------------------------------
                  mean    stddev      min       max
desc count *     38.65    304.02        2      9051
leaf count *     20.34    152.14        2      4539
child count*      2.00      0.06        2         4
path depth +   2561.52   3186.79        1     10001
node depth +     20.29      6.70        1        40
node depth #     21.26      6.58        1        40
fork depth #    152.67    121.75        1       694

* : stats over internal nodes (not the root)
+ : stats over all nodes (not the root)
# : stats over leaf nodes

Tree Level Expansion
-------------------------------------------------
level expansion ratio

1 1.000000
2 0.500000
3 0.218750
4 0.082031
5 0.031250
6 0.011475
7 0.004028
8 0.001358
9 0.000450
```

```
10 0.000140
11 0.000041
12 0.000012
13 0.000003
14 0.000001
.. ..
694 0.000000
```

20.3.6. Discretized Linear Congruential. Data was generated based on a primitive linear congruential generator:

$$x(t+1) = 37379 \cdot x(t) \mod 17203 \tag{20.4}$$

The real valued series $x(t)$ was discretized as 'a' if $x(t) \mod 3 = 0$, 'b' if $x(t) \mod 3 = 1$ and 'c' otherwise.

```
Standard Statistics
-------------------------------------------------
input length       10001
alphabet size      4
alphabet           abc~
nodes              18515
leafs              10001
internal nodes     8514

Alphabet Histogram
-------------------------------------------------
a                   3307
b                   3333
c                   3360
~                   1

Summary Statistics of Suffix Tree
-------------------------------------------------
                 mean    stddev      min       max
desc count *    16.53    140.86        2      6210
leaf count *     9.79     76.07        2      3360
child count*     2.17      0.38        2         4
path depth +  3194.72   3033.98        1     10001
node depth +     8.60      1.27        1        12
node depth #     9.33      0.80        1        12
fork depth #  1822.27   1363.89        1      4267

* : stats over internal nodes (not the root)
+ : stats over all nodes (not the root)
# : stats over leaf nodes

Tree Level Expansion
-------------------------------------------------
level expansion ratio

1 1.000000
2 0.625000
3 0.437500
4 0.320312
5 0.238281
```

```
6 0.178223
7 0.125549
8 0.060654
9 0.019634
10 0.005311
11 0.001354
12 0.000341
13 0.000085
14 0.000021
15 0.000005
16 0.000001
17 0.000000
.. ..
4267 0.000000
```

20.3.7. The "showcase" 2-dimensional HMM. Sample generated from showcase 2-dimensional OOM constructed by transforming a HMM working on the alphabet $\Sigma = \{a, b\}$. Defined via the equations 20.5 and 20.6.

$$\tau_a = \begin{pmatrix} 1/8 & 1/5 \\ 3/8 & 0 \end{pmatrix}, \quad \tau_b = \begin{pmatrix} 1/8 & 4/5 \\ 3/8 & 0 \end{pmatrix} \tag{20.5}$$

$$w_0 = (2/3, 1/3)^T \tag{20.6}$$

```
Standard Statistics
-------------------------------------------------
input length       10000
alphabet size      2
alphabet           ab
nodes              19963
leafs              9982
internal nodes     9981

Alphabet Histogram
-------------------------------------------------
a                   3360
b                   6640

Summary Statistics of Suffix Tree
-------------------------------------------------
                 mean    stddev      min       max
desc count *    25.35    220.19        2     13250
leaf count *    13.67    110.10        2      6626
child count*     2.00      0.00        2         2
path depth +  2511.75   3223.76        1     10000
node depth +    13.67      2.41        1        21
node depth #    14.67      1.96        1        21
fork depth #    16.09      2.73        7        31

* : stats over internal nodes (not the root)
+ : stats over all nodes (not the root)
```

```
# : stats over leaf nodes

Tree Level Expansion
-------------------------------------------------
level expansion ratio

1 1.000000
2 1.000000
3 1.000000
4 1.000000
5 1.000000
6 1.000000
7 1.000000
8 0.976562
9 0.939453
10 0.864258
11 0.751465
12 0.607422
13 0.451294
14 0.308533
15 0.194550
16 0.114822
17 0.064476
18 0.034668
19 0.018076
20 0.009248
21 0.004681
22 0.002356
23 0.001183
24 0.000593
25 0.000297
26 0.000149
27 0.000074
28 0.000037
29 0.000019
30 0.000009
31 0.000005
```

20.3.8. Independently Distributed.

```
Standard Statistics
-------------------------------------------------
input length       10001
alphabet size      4
alphabet           abc~
nodes              18534
leafs              10001
internal nodes     8533

Alphabet Histogram
-------------------------------------------------
a                   331
b                   630
c                   9039
~                   1
```

```
Summary Statistics of Suffix Tree
-------------------------------------------------
                  mean    stddev      min       max
desc count *     46.55    423.88        2     16749
leaf count *     25.99    228.96        2      9039
child count*      2.17      0.38        2         4
path depth +   2710.70   3262.98        1     10001
node depth +     22.43     10.09        1        58
node depth #     23.17     10.06        1        58
fork depth #     29.02     11.73        1        67

* : stats over internal nodes (not the root)
+ : stats over all nodes (not the root)
# : stats over leaf nodes

Tree Level Expansion
-------------------------------------------------
level   expansion ratio

1       1.000000
2       0.625000
3       0.375000
4       0.210938
5       0.099609
6       0.041504
7       0.015991
8       0.005798
9       0.001976
10      0.000657
11      0.000210
12      0.000065
13      0.000020
14      0.000006
15      0.000002
16      0.000001
..      ..
67      0.000000
```

CHAPTER 21

Future Development

In this section I will shortly given some thoughts where OMK could improve in the immediate future. I think OMK should strive for two overall goals

(1) integrated coherent package
(2) simplified usage

21.1. Integrated Coherent Package

For making OMK into an integrated, coherent package, two subgoals must be addressed:

(1) fully integrate the OOM estimation method from [**Kre**] with OMK
(2) provide a set of small, self-contained and simple to use command line tools that can be combined (like Unix commands) for
  (a) suffix tree, context graph building and analysis (both character and word based)
  (b) information theoretic sample analysis
  (c) OOM run-mode
  (d) OOM estimation: greedycluster, softcluster

The tools should use a simple external file format for storing OOMs like e.g. Matlab scriptcode (use a Matlab ASCII format to write/read OOM definitions) or XML. The functionality is already there, but lacks integration, coherence and polish to make a full set of user level tools.

A final step would be to assemble a ready to use binary distribution of OMK for popular platforms.

21.2. Simplified Usage

Simplified usage can apply to at least two different levels of usage: end-user and developer. The simplification for the end-user will likely come as a result of easy to use command line tools and a ready to use binary distribution. This was discussed in the last subsection. The simplification for the developer using OMK to build new applications will come from

(1) simple and consistent package interfaces
(2) small codebase to understand
(3) good package documentation and examples
(4) limited external dependencies

Currently, the OMK has the following external dependencies: Standard C++ library and BLAS/LAPACK. The softclustering method has the additional dependency on PORT due to the use of constrained non-linear optimization. The sftree command line tool has the dependency on *GNU getopt* for command line argument parsing. The PORT library is free to use only for research purposes but the dependency is likely unavoidable at this

time. *GNU getopt* use should be surrected in the future. It is not worth the dependency it introduces.

Even during the short time of OMK development, the package has seen a growth to approximately 30.000 lines of code. A certain amount of this codebase was accumulated during an experimental phase and is no longer in active use. Throwing out code no longer used and refactoring the package to reduce the overall amount of code is thus a valid target.

21.3. Long-term

On the long-term, when a very good learning method is established and fully developed a *pure ANSI-C* implementation without any external dependencies, no BLAS/LAPACK, no whatsoever could be targeted.

The main reason for this perspective is, that though a pure ANSI-C implementation would be inferior from a software engineering stand, such a package implementation would likely ensure widest acceptance and spread.

Bibliography

[Ale01] Andrei Alexandrescu, *Modern C++ Design*, Addison Wesley, 2001.

[Apo85] A. Apostolico, *The myriad virtues of subword trees*, Combinatorial Algorithms on Words, NATO ISI Series, Springer, 1985, pp. 85–96.

[BK00] Bernhard Balkenhol and Stefan Kurtz, *Universal Data Compression Based on the Burrows and Wheeler Transformation: Theory and Practice.*, IEEE Transactions on Computers **49** (2000), no. 10, 1043–1053.

[BRY98] A.R. Barron, J. Rissanen, and B. Yu, *The MDL principle in modeling and coding*, IEEE Trans. Inform. Theory **44** (1998), 2743–2760, Special issue in IEEE - Commemorating 50 years of information theory.

[CT91] Thomas M. Cover and Joy A. Thomas, *Elements of Information Theory*, Wiley Series in Telecommunications, Wiley, 1991.

[Die96] Reinhard Diestel, *Graphentheorie*, Springer, 1996.

[Far96] Martin Farach, *Optimal Suffix Tree Construction with Large Alphabets*, Proceedings of the 38th Annual Symposium on the Foundations of Computer Science, IEEE Comput. Soc. Press, 1996.

[FFM] M. Farach, P. Ferragina, and S. Muthukrishnan.

[Fre60] E. Fredkin, *Trie Memory*, Comm. ACM **3** (1960), no. 9, 490–499.

[Gal68] R.G. Gallager, *Information Theory and Reliable Communication*, Wiley, New York, 1968, (Theorem 3.5.3).

[GK97] Robert Giegerich and Stefan Kurtz, *From Ukkonen to McCreight and Weiner: A Unifying View of Linear-Time Suffix Tree Construction*, Algorithmica **19** (1997), no. 3, 331–353.

[GKS99] Robert Giegerich, Stefan Kurtz, and Jens Stoye, *Efficient Implementation of Lazy Suffix Trees*, Proc. of 3rd Workshop on Algorithmic Engineering, July 1999.

[Gra90] Robert M. Gray, *Entropy and Information Theory*, Springer-Verlag, Information Systems Laboratory, Stanford University, 1990.

[Gro] Object Management Group, *The Unified Modeling Language (UML)*, {`http://www.omg.org/technology/documents/formal/uml.htm`}.

[Gus97] D. Gusfield, *Algorithms on Strings, Trees, and Sequences*, Cambridge University Press, 1997.

[HU79] John E. Hopcraft and Jeffrey Ullman, *Introduction to automate theory, languages and computation*, Addison-Wesley, 1979.

[IHS+01a] Shunsuke Inenaga, Hiromasa Hoshino, Ayumi Shinohara, Masayuki Takeda, and Setsuo Arikawa, *On-Line Construction of Symmetric Compact Directed Acyclic Word Graphs*, Tech. Report DOI-TR-CS-183, Department of Informatics, Kyushu University, Fukuoka (Japan) and PRESTO, Japan Science and Technology Corporation (JST), January 2001.

[IHS+01b] Shunsuke Inenaga, Hiromasa Hoshino, Ayumi Shinohara, Masayuki Takeda, Setsuo Arikawa, Giancarlo Mauri, and Giulio Pavesi, *On-Line Construction of Compact Directed Acyclic Word Graphs*, Combinatorial Pattern Matching, 2001, pp. 169–180.

[int] *Intel Math Kernel Library (MKL)*, {`http://www.intel.com/software/products/mkl/mkl52/index.htm`}.

[Jae98] Herbert Jaeger, *Discrete-time, Discrete-valued Observable Operator Models: a Tutorial*, Tech. Report 42, GMD, Sankt Augustin, 1998.

[Jae99] ———, *Characterizing distributions of stochastic processes by linear operators*, Tech. Report 62, GMD, Sankt Augustin, 1999.

[Jae00] ———, *Observable Operator Models for Discrete Stochastic Time Series*, Neural Computation **12** (2000), no. 6, 1371–1398.

[Jae01] ———, *Modeling and learning continuous-valued stochastic processes with OOMs*, Tech. Report 102, GMD, Sankt Augustin, 2001.

[JR85] B.H. Juang and L.R. Rabiner, *A Probabilistic Distance Measure for Hidden Markov Models*, AT&T Technical Journal **64** (1985), no. 2.

[Kre] Klaus Kretzschmar, *private communication*, {`http://www.ais.fraunhofer.de/INDY/klaus/`}.

[Lar98] N. J. Larsson, *The context trees of block sorting compression*, Proceedings of the IEEE Data Compression Conference, March 1998, pp. 189–198.

[LEN02] C. Leslie, E. Eskin, and W.S. Noble, *The Spectrum Kernel: A String Kernel for SVM Proteing Classification*, Pacific Symposium on Biocomputing 2002, vol. 7, January 2002, pp. 566–575.

[Maa00] Moritz G. Maass, *Linear bidirectional on-line construction of affix trees*, Proc. 11th Ann. Symp. on Combinatorial Pattern Matching, Springer, 2000, pp. 320–334.

[McC76] Edward M. McCreight, *A Space-Economical Suffix Tree Construction Algorithm*, Journal of the ACM **23** (1976), no. 2, 262–272.

[MN98] M. Matsumoto and T. Nishimura, *Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator*, ACM Transactions on Modeling and Computer Simulation **8** (1998), no. 1, 3–30.

[Mor68] D. R. Morrison, *PATRICIA - Practical Algorithm to Retrieve Information Coded Alphanumeric*, Journal of the ACM **15** (1968), no. 4, 514–534.

[Mus] David R. Musser, *Generic Programming*.

[net] *LAPACK - Linear Algebra PACKage*, {`http://www.netlib.org/lapack`}.

[Oes01] Bernd Oestereich, *Objektorientierte Softwareentwicklung - Analyse und Design mit der Unified Modeling Language*, Oldenburg, 2001.

[por] *PORT Mathematical Subroutine Library*, {`http://www.bell-labs.com/project/PORT`}.

[PSLM01] P.J. Plauger, Alexander A. Stepanov, Meng Lee, and David R. Musser, *The C++ Standard Template Library*, Prentice Hall PTR, 2001.

[Shi95] A.N. Shiryaev, *Probability*, 2nd Ed. ed., Graduate Texts in Mathematics, vol. 95, Springer, 1995.

[Sin92] Yakov G. Sinai, *Probability Theory - an Introductory Course*, Springer, 1992.

[Sto79] Josef Stoer, *Einfüehrung in die Numerische Mathematik I*, 2nd edition ed., Springer, 1979.

[Sto95] J. Stoye, *Affixbäume*, Master's thesis, Universität Bielefeld, May 1995.

[Str00] Bjarne Stroustrup, *The C++ Programming Language*, Special Edition ed., Addison Wesley, 2000.

[Szp93a] Wojciech Szpankowski, *A generalized suffix tree and its (un)expected asymptotic behaviors*, SIAM J. Comp. (1993), no. 22, 1176–1198.

[Szp93b] ———, *Asymptotic Properties of Data Compression and Suffix Trees*, IEEE Transactions on Information Theory **39** (1993).

[Ukk95] E. Ukkonen, *On-line construction of suffix trees*, Algorithmica (1995), no. 14, 249–260.

[Vap98] Vladimir N. Vapnik, *Statistical Learning Theory*, John Wiley and Sons, 1998.

[Vap99] ———, *An Overview of Statistical Learning Theory*, IEEE Trans. on Neural Networks **10** (1999), no. 5, 988–999.

[Wei73] P. Weiner, *Linear pattern matching algorithms*, Proceedings of the 14th Symposium on Switching and Automata Theory, 1973, pp. 1–11.

[Wyn95] Aaron D. Wyner, *1994 Shannon Lecture - Typical Sequences and All That: Entropy, Pattern Matching, and Data Compression*, IEEE Information Theory Society Newsletter (1995).