Université catholique de Louvain
Faculté des sciences appliquées
Département d'Ingénierie Informatique

# First Passage Times Dynamics in Markov Models with Applications to HMM Induction, Sequence Classification and Graph Mining

Jérôme Callut

Thèse présentée en vue de l'obtention du
grade de Docteur en Sciences Appliquées

Membres du Jury:

Prof. **Gianluca Bontempi** (ULB-Département d'Informatique)
Prof. **Philippe Delsarte** (UCL-Département d'Ingénierie Informatique)
Prof. **Yves Deville** (UCL-Département d'Ingénierie Informatique), Président
Prof. **Pierre Dupont** (UCL-Département d'Ingénierie Informatique), Promoteur
Prof. **Marco Saerens** (UCL IAG, Unité de Systèmes d'Information)
Prof. **Marc Sebban** (Université de Saint-Etienne, Laboratoire Hubert Curien, France)

Octobre 2007

# Acknowledgements

I would like to thank Prof. Pierre Dupont for introducing me to the field of Grammatical Inference and to Hidden Markov Models; supervising me closely while letting me free in the choices for this research; sharing with me his expertise in the presentation of results, both verbally and in writing; his precious proofreadings and suggestions in the drafting of this thesis and last but not least the good times we had together at conferences and in the department.

I am grateful to Prof. Marco Saerens for recommending me, throughout my research, excellent and useful publications on the theory of Markov chains; the exciting debates we had on the various implications of Machine Learning; his enthusiasm for research that still stimulates me to continue in this field.

Prof. Philippe Delsarte is a well of erudition for a researcher like me who has to make a daily use of mathematical tools such as linear algebra. I would like to thank him for his remarkable availability; the numerous and exciting afternoons we spent in his office solving problems related to my research or otherwise; encouraging me to go off the beaten tracks in order to broaden my scientific culture.

I am grateful to Prof. Gianluca Bontempi for helping me in taking some distance from my research and in developing a well-informed point of view on the different results I obtained; advising me on how to make this thesis as understandable as possible for lay readers .

I would like to thank Prof. Guy Latouche for his precious teaching of stochastic models during my studies; his expert advice concerning phase-type distributions and helping me to solve technical problems in this field.

I am grateful to Prof. Yves Deville for the interesting debates about optimization problems during our research meetings and his interesting comments and suggestions concerning my research.

I would like to thank Prof. Marc Sebban for his very careful proofreading, his remarks and suggestions to improve the present document and the nice discussions we had about research or otherwise.

I would like to thank colleagues sharing my office: Stéphane Zampelli, Grégoire Dooms and Thibault Helleputte for the good atmosphere at work and outside work; the emulation that working with them has provided; their sympathy and constant support while drafting this thesis.

Thanks also go to my other colleagues, and more particularly to Raphaël Collet, Renaud De Landsheer, Pierre Schaus, Jean-Noël Monette and Sébastien Gonzalez for the good atmosphere in the department, the scientific – or other – debates, and their support.

I am grateful to Natacha Wittorski for her constant support throughout my research and especially at the end when times were getting harder; her everyday care which made life easier for me and enabled me to concentrate on my research.

I would like to thank my family and more particularly my parents Françoise Remy and Jean-Paul Callut for always allowing me to make my choices freely; their support and encouragements throughout my studies and my research.

I am grateful to my uncle Jean-Pierre Callut for his precious proofreadings and encouragements while drafting this thesis.

Finally, I would like to thank my friends and more particularly Boris Verdeyen, Agustin Eguia, Laurent Blasson, Cédric Blasson and Philip Catherine for supporting me and helping me restore self confidence in hard times.

## Abstract

Sequential data are encountered in many contexts of everyday life and in numerous scientific applications. They can for instance be SMS typeset on mobile phones, web pages reached while crossing hyperlinks, system logs or DNA samples, to name a few. Generating such data defines a *sequential process*.

This thesis is concerned with the modeling of sequential processes from observed data. Sequential processes are here modeled using probabilistic models, namely *discrete time Markov chains* (MC), *Hidden Markov Models* (HMMs) and *Partially Observable Markov Models* (POMMs). Such models can answer questions such as (i) *Which* event will occur a couple of steps later? (ii) *How many times* will a particular event occur? and (iii) *When* does an event occur for the first time given the current situation?

The last question is of particular interest in this thesis and is mathematically formalized under the notion of *First Passage Times (FPT) dynamics* of a process. The FPT dynamics is used here to solve the three following problems related to *machine learning* and *data mining*: (i) HMM/POMM induction, (ii) supervised sequence classification and (iii) relevant subgraph mining.

Firstly, we propose a novel algorithm, called POMMStruct, for learning the structure and the parameters of POMMs to best fit the empirical FPT dynamics observed in the samples. Experimental results illustrate the benefit of POMMStruct in the modeling of sequential processes with a complex temporal dynamics while compared to classical induction approaches. Our second contribution is concerned with the classification of sequences. We propose to model the FPT in sequences with discrete *phase-type* (PH) distributions using a novel algorithm called PHit. These distributions are used to devise a new *string kernel* and a probabilistic classifier. Experimental results on biological data shows that our methods provides state-of-the-art classification results. Finally, we address the problem of mining subgraphs, which are relevant to model the relationships between selected nodes of interest, in large graphs such as biological networks. A new relevance criterion based on particular random walks called *K-walks* is proposed as well as efficient algorithms to compute this criterion. Experiments on the KEGG metabolic network and on randomly generated graphs are presented.

# Frequently used symbols

## Matrices

| | |
|---|---|
| $(I - F)^{-1}$ | The fundamental matrix of an absorbing MC |
| $A$ | The transition probability matrix of a MC |
| ${}^{x}A$ | The transition matrix used for $\mathcal{K}$-walks starting in state $x$ |
| $B$ | The emission matrix of a HMM |
| $E$ | The transition submatrix between transient states and absorbing states in an absorbing MC |
| ${}^{x}E$ | The absorption vector used for $\mathcal{K}$-walks starting in state $x$ |
| $F$ | The transition submatrix between transient states |
| ${}^{x}F$ | The transient submatrix used for $\mathcal{K}$-walks starting in state $x$ |
| $I$ | The identity matrix |
| $K$ | A positive definite Gram or kernel matrix |
| $M$ | The adjacency matrix of a graph |
| ${}^{x}N$ | The fundamental matrix used for $\mathcal{K}$-walks starting in state $x$ |

## Vectors

| | |
|---|---|
| $\mathbf{0}$ | A vector having all its elements equal to 0 |
| $\mathbf{1}$ | A vector having all its elements equal to 1 |
| $\boldsymbol{\alpha}$ | The vector of Lagrange multipliers or dual variables |
| $\boldsymbol{\theta}$ | The parameters of a multinomial distribution |
| $\boldsymbol{\iota}$ | The initial distribution vector of a MC |
| $\boldsymbol{\xi}$ | The vector of slack variables in the soft-margin SVM |
| $\boldsymbol{\pi}$ | The stationary distribution of an irreducible MC |
| $\boldsymbol{\sigma}^{\kappa_a}$ | An initial distribution relative to the POMM block $\kappa_{\mathtt{a}}$ |
| $\boldsymbol{\tau}$ | A vector such that $\tau_{q_0} = \mathrm{mta}(q_0)$ |
| $\boldsymbol{\psi}$ | The parameters of a Dirichlet prior distribution |
| $\boldsymbol{c}$ | A vector of counts or weights |
| $\boldsymbol{e}$ | The absorption vector of a reduced absorbing MC |
| $\boldsymbol{r}_i, \boldsymbol{l}_i$ | The right and left eigenvectors associated to $\lambda_i$ |
| $\boldsymbol{u}$ | The initial distribution vector for transient states |
| $\boldsymbol{w}$ | The weight vector of a hyperplane |
| $\boldsymbol{x}, \boldsymbol{x}_i$ | Instances or a data points (not necessarily a vector) |

**Scalars**

| | |
|---|---|
| $\alpha, \alpha_i$ | Dual Lagrange multipliers or variables |
| $\epsilon$ | A precision parameter for convergence checking |
| $\theta_i$ | A parameter of a multinomial distribution |
| $\lambda_i$ | The $i$-th largest eigenvalue of a matrix |
| $\lambda_{max}$ | The largest eigenvalue of a matrix |
| $\mu$ | A precision parameter in state merging |
| $\xi, \xi_i$ | Slack variables in SVM |
| $b$ | The bias term of a hyperplane |
| $C$ | The regularization constant in SVM |
| $c, c_i$ | Counts or a weights |
| $d^{opt}$ | The optimal solution of an optimization problem in dual form |
| $h_{VC}$ | The VC dimension of a class of functions |
| $L_{max}$ | A maximum walk length |
| $n$ | The number of instances in the training set |
| $N$ | A subsequence length in $N$-grams and FPT features |
| $n_t$ | The number of transitions/edges in a model |
| $n_s$ | The number of states/nodes in a model |
| $p^{opt}$ | The optimal solution of an optimization problem in primal form |
| $q, q', q_1, q_2$ | States in Markov models or in probabilistic automata |
| $q_0, q_f$ | The initial and final states (possibly silent) |
| $x$ | A node of interest |
| $y, y_i$ | Classes or the labels of instances |
| $z, z_i$ | Discrete passage times |

**Sequences**

| | |
|---|---|
| $\|s\|$ | The length of the sequence $s$ |
| $\varepsilon$ | The empty sequence |
| $\mathtt{a}, \mathtt{b}, \mathtt{c} \in \Sigma$ | Symbols in the alphabet $\Sigma$ |
| $h$ | A sequence of past events (*history*) |
| $l$ | The last position or index in a sequence |
| $s, s', s^i$ | Discrete sequences |
| $s_{i:j}$ | The subsequence of $s$ from position $i$ to position $j$ |
| $s_{:j}$ | The subsequence of $s$ starting at position 0 and ending at position $j$ |
| $s_{i:}$ | The subsequence of $s$ starting at position $i$ and ending at position $l$ |
| $v, v^1, v^2$ | Subsequences |

**Sets and partitions**

| | |
|---|---|
| $\kappa$ | A state partition in POMMs |
| $\kappa_{\mathtt{a}}$ | A block gathering states emitting symbol $\mathtt{a}$ |
| $\Sigma$ | A discrete alphabet of symbols |
| $\rho$ | The probabilistic parameters of a model |
| $\Omega$ | The set of all possible discrete events |
| $\mathcal{C}$ | An equivalence class of states in a MC |
| $\mathcal{E}$ | The set of edges of a graph |
| $\mathcal{F}$ | A class of functions |
| $\mathcal{G}$ | A Hilbert space (a.k.a the feature space) |
| $\mathcal{H}$ | A set of hidden paths |
| $\mathcal{H}^{\mathrm{a,b}}$ | A set of hidden paths from block $\kappa_{\mathtt{a}}$ to block $\kappa_{\mathtt{b}}$ |
| $\mathcal{K}$ | A set of nodes of interest |
| $\mathcal{M}$ | A set of generative models |
| $\mathcal{P}$ | A set of features consisting of subsequence pairs |
| $Q$ | The set of states of a MC |
| $Q_A$ | The set of absorbing states of an absorbing MC |
| $Q_T$ | The set of transient states of an absorbing MC |
| $S/S_{\mathrm{test}}$ | A sample of training/test sequences |
| $Tr$ | A training set made of labeled sequences |
| $\mathcal{V}$ | The set of nodes of a graph |
| $\mathcal{W}$ | The set of $\mathcal{K}$-walks |
| $\mathcal{W}_L$ | The set of $\mathcal{K}$-walks of length $L$ |
| $\mathcal{W}_{L_{max}}$ | The set of $\mathcal{K}$-walks of length up to $L_{max}$ starting in $x$ |
| ${}^{x}\mathcal{W}$ | The set of $\mathcal{K}$-walks starting in state $x$ |
| ${}^{x}\mathcal{W}_L$ | The set of $\mathcal{K}$-walks of length $L$ starting in $x$ |
| ${}^{x}\mathcal{W}_{L_{max}}$ | The set of $\mathcal{K}$-walks of length up to $L_{max}$ starting in $x$ |
| $\mathcal{X}$ | The input space where live the instances |
| $\mathcal{Y}$ | The set of labels or classes |
| $Z$ | A set of first passage times |

**Probabilistic functions, distributions and random variables**

| | |
|---|---|
| $\psi(\boldsymbol{\theta})$ | The Dirichlet prior parameters relative to $\boldsymbol{\theta}$ |
| $D_{JS}(P_1, P_2)$ | The Jensen-Shannon divergence between $P_1$ and $P_2$ |
| $D_{JS}^{\mathrm{gen}}(P_1, \ldots, P_r)$ | The generalized Jensen-Shannon divergence between $P_1, \ldots, P_r$ |
| $D_{KL}(P_1, P_2)$ | The Kullback-Leibler divergence between $P_1$ and $P_2$ |
| $\mathbb{E}[X]$ | The expectation of the random variable $X$ |
| $H(P)$ | The Shannon entropy of distribution $P$ |
| $\mathbb{1}\{X = x\}$ | An indicator variable equal to 1 if $X = x$ and to 0 otherwise |
| $\mathcal{P}_{\boldsymbol{\theta}}$ | A multinomial distribution parametrized by $\boldsymbol{\theta}$ |

### Sequence related functions

| | |
|---|---|
| $C(h)$ | The number of times $h$ occurs in $S$ |
| $C(h, \mathtt{a})$ | The number of times $h$ followed by $\mathtt{a}$ occurs in $S$ |
| $C^y(h)$ | The number of times $h$ occurs in sequences labeled with $y$ |
| $\text{count}(\mathtt{a}, \mathtt{b})$ | the number of times $\mathtt{a}$ is followed by $\mathtt{b}$ in $S$ |
| $\text{count}(v^1, v^2)$ | the number of times $v^1$ is followed by $v^2$ in $S$ |
| $\text{FPT}(\mathtt{a}, \mathtt{b})$ | The FPT from $\mathtt{a}$ to $\mathtt{b}$ in $S$ |
| $\widehat{\text{FPT}}(\mathtt{a}, \mathtt{b})$ | The empirical FPT distribution from $\mathtt{a}$ to $\mathtt{b}$ in $S$ |
| $\text{FPT}_s(\mathtt{a}, \mathtt{b})$ | The FPT from $\mathtt{a}$ to $\mathtt{b}$ in sequence $s$ |
| $\text{FPT}_s(v^1, v^2)$ | The FPT from $v^1$ to $v^2$ in sequence $s$ |
| $\widehat{\text{FPT}}_{S_{\text{test}}}(\mathtt{a}, \mathtt{b})$ | The empirical FPT distribution from $\mathtt{a}$ to $\mathtt{b}$ in $S_{\text{test}}$ |
| $\text{FPT}_y(v^1, v^2)$ | The FPT from $v^1$ to $v^2$ in class $y$ |
| $\widehat{\text{FPT}}_y(v^1, v^2)$ | The empirical FPT distribution from $v^1$ to $v^2$ in class $y$ |

### Markov Chain related functions and relations

| | |
|---|---|
| $\alpha^{\varphi}(q, t)$ | The PHit forward variables |
| $\alpha^{K}(q, l)$ | The limited $\mathcal{K}$-walks forward variables |
| $\beta^{\varphi}(q, t)$ | The PHit backward variables |
| $\beta^{\mathcal{K}}(q, l)$ | The limited $\mathcal{K}$-walks backward variables |
| $\varphi$ | A discrete phase-type distribution |
| $\varphi^{y}_{(v^1, v^2)}(.)$ | The PH distribution associated to $(v^1, v^2)$ estimated from class $y$ |
| $er_{\mathcal{K}}(q, q')$ | The edge relevance of $q \to q'$ |
| $\text{fpt}(q_0, q)$ | The first passage time from $q_0$ to $q$ |
| $\text{mfpt}(q_0, q)$ | The mean first passage time from $q_0$ to $q$ |
| $\text{mpt}(q_0, q)$ | The mean passage time in $q$, starting in $q_0$ |
| $\text{mta}(q_0)$ | The mean time to absorption, starting in $q_0$ |
| $N^{\varphi}(q, q')$ | The number of times the process triggers $q \to q'$ |
| $nr_{\mathcal{K}}(q)$ | The node relevance of $q$ |
| $P_{\text{FPT}}[X_0 \mid X_0 \in \kappa_{\mathtt{a}}]$ | The initial distribution for FPT starting in block $\kappa_{\mathtt{a}}$ |
| $P_{trap}[\mathcal{C}_i]$ | The absorption probability in final class $\mathcal{C}_i$ |
| $P_{trap}[q]$ | The absorption probability in state $q$ |
| $\text{pt}(q_0, q)$ | The passage time in $q$, starting in $q_0$ |
| $q \rightsquigarrow q'$ | The state $q$ leads stochastically to the state $q'$ |
| $S^{\varphi}(q)$ | The number of times the process starts in state $q$ |
| $\text{ta}(q_0)$ | The time to absorption, starting in $q_0$ |
| $X_t$ | The state reached by a MC at time $t$ |

**HMM related functions**

| | |
|---|---|
| $\alpha(q,t)$ | The standard forward variables for HMMs |
| $\beta(q,t)$ | The standard backward variables for HMMs |
| $\iota(q)$ | The initial function of a HMM |
| $A(q,q')$ | The transition function of a HMM |
| $B(q,\mathtt{a})$ | The emission function of a HMM |
| $N(q,q')$ | The number of transitions from state $q$ to state $q'$ |
| $n_e(q)$ | The number of symbols state $q$ emits with a positive probability |
| $n_t(q)$ | The number of transitions outgoing from state $q$ |
| $O_t$ | The symbol emitted by the emission process at time $t$ |
| $P_H(s)$ | The probability of a sequence $s$ w.r.t. the HMM $H$ |
| $P_H(s,\nu)$ | The probability of a sequence $s$ along path $\nu$ in the HMM $H$ |
| $S(q)$ | The number of times the process starts in state $q$ |
| $V(q,\mathtt{a})$ | The number of times the process reaches state $q$ and emits $\mathtt{a}$ |

**Probabilistic automaton related functions**

| | |
|---|---|
| $\delta(q,\mathtt{a})$ | The unique successor of $q$ following transition $\mathtt{a}$ in a PDFA |
| $\phi(q,\mathtt{a})$ | The deterministic transition function of a PDFA |
| $\phi(q,\mathtt{a},q')$ | The transition function of a PNFA |
| $C(q)$ | The number of times state $q$ is used while generating the sample |

**Partially Observable Markov Model related functions**

| | |
|---|---|
| $\mathrm{fpt}(\mathtt{a},\mathtt{b})$ | The first passage time from block $\kappa_\mathtt{a}$ to block $\kappa_\mathtt{a}$ |
| $\alpha^{\mathtt{a},\mathtt{b}}(q,t)$ | The POMMPHIT forward variables |
| $\beta^{\mathtt{b}}(q,t)$ | The POMMPHIT backward variables |
| $S^{\mathtt{a},\mathtt{b}}(q)$ | The number of times the process starts in state $q \in \kappa_\mathtt{a}$ during FPT from $\kappa_\mathtt{a}$ to $\kappa_\mathtt{b}$ |
| $N^{\mathtt{a},\mathtt{b}}(q,q')$ | The number of times the process triggers $q \to q'$ during FPT from $\kappa_\mathtt{a}$ to $\kappa_\mathtt{b}$ |

**Support Vector Machine related functions**

| | |
|---|---|
| $\langle \boldsymbol{x}, \boldsymbol{z} \rangle$ | The dot product of $\boldsymbol{x}$ and $\boldsymbol{z}$ |
| $\Phi(\boldsymbol{x})$ | The (implicit) Hilbert space mapping of a kernel |
| $dim(\mathcal{X})$ | The dimension of the input space $\mathcal{X}$ |
| $f(\boldsymbol{x})$ | The decision function in classification problems |
| $f^{opt}(\boldsymbol{x})$ | The decision function having the lowest functional risk |
| $g(\boldsymbol{x})$ | The linear function implemented by a SVM |
| $k(\boldsymbol{x}, \boldsymbol{z})$ | A positive definite kernel function |
| $L(.)$ | The Lagrangian of a constrained optimization problem |
| $R[f]$ | The functional risk in the statistical learning theory |
| $R_{emp}[f]$ | The empirical risk in the statistical learning theory |
| $R_{reg}[f]$ | The regularizedl risk in the statistical learning theory |
| $sgn(.)$ | The function returning the sign (-1/+1) of a scalar |
| $W^{\mathrm{dl}}(\ldots)$ | The dual objective function |
| $W^{\mathrm{pr}}(\ldots)$ | The primal objective function |

# Table of acronyms

| | |
|---|---|
| AIC | Akaike's Criterion |
| BIC | Bayesian Information Criterion |
| CLL | Conditional Log Likelihood |
| CRF | Conditional Random Field |
| CP | Convex Optimization Problem |
| DA | Deterministic Annealing |
| DFA | Deterministic Finite state Automaton |
| EM | Expectation-Maximization |
| FPT | First Passage Time(s) |
| GCD | Greatest Common Divisor |
| HMM | Hidden Markov Model |
| HMMT | HMM with emission on transitions |
| HMSVM | Hidden Markov SVM |
| IPM | Interior Point Method |
| JS | Jensen-Shannon |
| KKT | Karush-Kuhn-Tucker (optimality conditions) |
| KL | Kullback-Leibler |
| KN | Kneser-Ney (back-off scheme) |
| LRU | Least Recently Used |
| MA | Multiplicity Automaton |
| MAP | Maximum *A Posteriori* |
| MC | Discrete Time Markov Chain |
| MCE | Minimum Classification Error |
| MCL | Markov Cluster Algorithm |
| MDL | Minimum Description Length |
| MEMM | Maximum Entropy Markov Models |
| MFPT | Mean First Passage Time(s) |
| ML | Maximum Likelihood |
| MLG | Machine Learning Group |
| MLP | Multiple Layer Perceptron |
| MMI | Maximum Mutual Information |
| NER | Named Entity Recognition |
| NFA | Non-deterministic Finite state Automaton |
| OP | Optimization Problem |
| PA | Probabilistic automaton/automata |
| PAC | Probably Approximately Correct |
| PDFA | Probabilistic Deterministic Finite state automata |
| PNFA | Probabilistic Non-derterministic Finite state automata |
| PH | Discrete Phase-type distribution |
| POMM | Partially Observable Markov Model |
| POS | Part-Of-Speech |
| PPTA | Probabilistic Prefix Tree Automaton |

| | |
|---|---|
| PSRL | Pseudo-Stochastic Rational Language |
| QP | Quadratic Optimization Problem |
| RSL | Rational Stochastic Language |
| SDP | Semidefinite Programming |
| SMO | Sequential Minimal Optimization |
| SRM | Structural Risk Minimization |
| SVM | Support Vector Machine(s) |
| SV | Support Vector |
| VC | Vapnik-Chervonenkis |

# Contents

**Introduction** **1**

**I Background** **9**

**1 Discrete time Markov chains** **11**
1.1 Definitions . . . . . 12
1.2 The Perron-Frobenius theorem . . . . . 17
1.3 Dynamics of random walks . . . . . 18
1.4 Discrete Phase-type distributions . . . . . 25
1.5 Estimation techniques . . . . . 27

**2 Hidden Markov Models** **31**
2.1 Definitions . . . . . 32
2.2 Links between PA and HMMs . . . . . 35
2.3 The three main problems and their classical solutions . . . . 39
2.3.1 Computing the sequence likelihood . . . . . 40
2.3.2 Computing the optimal path . . . . . 42
2.3.3 Estimating the model parameters . . . . . 44

**3 Kernel methods** **47**
3.1 Statistical learning theory . . . . . 48
3.1.1 The functional and empirical risks . . . . . 48
3.1.2 Capacity, VC dimension and VC bounds . . . . . 49
3.1.3 Structural risk minimization and regularized risk . . . 51
3.2 Kernels and positive definite matrices . . . . . 51
3.3 Support Vector Machines (SVM) . . . . . 55
3.3.1 Definitions . . . . . 55
3.3.2 Primal and dual formulation . . . . . 57
3.3.3 Soft-margin SVM . . . . . 61
3.3.4 Relation with statistical learning theory . . . . . 64
3.3.5 Implementation . . . . . 67

**II Hidden Markov Model induction** **71**

**4 An overview of HMM induction techniques** **73**
4.1 Structural induction by parameter estimation . . . . . 74
4.1.1 Baum-Welch on fully connected graphs . . . . . 75
4.1.2 MAP estimation with entropic priors . . . . . 77
4.2 State merging and splitting . . . . . 79
4.2.1 The ALERGIA and MDI algorithms . . . . . 80
4.2.2 Bayesian state merging . . . . . 84
4.2.3 Maximum likelihood state splitting . . . . . 88
4.3 Discriminative induction techniques . . . . . 90
4.3.1 Conditional likelihood based techniques . . . . . 91
4.3.2 Margin-based techniques . . . . . 92
4.4 Summary and discussion . . . . . 93

**5 Learning HMMs from First Passage Times** **99**
5.1 First Passage Times in sequences . . . . . 100
5.2 Partially Observable Markov Models . . . . . 101
5.2.1 Equivalence between POMMs and HMMs . . . . . 102
5.2.2 Links between POMMS and Markov chains . . . . . 103
5.3 First Passage Times in POMMs . . . . . 107
5.4 Modeling long-term probabilistic dependencies . . . . . 110
5.4.1 Modeling long-term dependencies with finite order MC 111
5.4.2 Topology matters to fit long-term dependencies . . . . 112
5.4.3 Long-term dependencies and FPT . . . . . 114
5.5 Induction algorithm : POMMStruct . . . . . 116
5.5.1 Induction scheme . . . . . 116
5.5.2 Fitting the FPT: POMMPHit . . . . . 122
5.6 Experiments . . . . . 126
5.6.1 Datasets . . . . . 126
5.6.2 Evaluation criteria . . . . . 128
5.6.3 Evaluation of the model structure . . . . . 129
5.6.4 Complex dynamics modeling . . . . . 130
5.6.5 Influence of the parameters . . . . . 132
5.6.6 Comparative results . . . . . 136
5.6.7 Implementation issues . . . . . 144
5.7 Summary and discussion . . . . . 145

**III Sequence classification** **149**

**6 An overview of sequence classification methods** **151**
6.1 Generative approaches . . . 152
6.1.1 Naive Bayes and N-grams classifiers . . . 153
6.1.2 A compression-based approach . . . 154
6.2 Discriminative approaches . . . 157
6.2.1 Discriminative Markov models . . . 157
6.2.2 Kernel methods for sequence classification . . . 159
6.3 Summary and discussion . . . 167

**7 Sequence classification using First Passage Times** **173**
7.1 Classification using first passage times . . . 175
7.2 Feature extraction and selection . . . 175
7.3 Fitting PH distributions . . . 178
7.3.1 PHit algorithm . . . 179
7.3.2 Influence of the number of phases . . . 183
7.4 Building PH-based classifiers . . . 185
7.4.1 Maximum a posteriori classifier . . . 185
7.4.2 Phase-type kernel . . . 186
7.5 Experiments . . . 188
7.5.1 Dataset descriptions . . . 188
7.5.2 Influence of parameters . . . 189
7.5.3 Comparative results . . . 192
7.5.4 Implementation issues . . . 192
7.6 Summary and discussion . . . 195

**IV Graph mining** **199**

**8 Relevant subgraph mining from First Passage Times** **201**
8.1 Overview and intuition . . . 203
8.2 $\mathcal{K}$-walks in a graph . . . 210
8.2.1 A dynamic view of the graph . . . 211
8.2.2 Relevance computation . . . 213
8.3 Limited $\mathcal{K}$-walks in a graph . . . 215
8.3.1 Walk and relevance definitions . . . 215
8.3.2 Links between standard and limited $\mathcal{K}$-walks . . . 217
8.3.3 Efficient relevance computation . . . 220
8.4 Groups of nodes of interest . . . 221
8.5 Subgraph extraction . . . 222
8.6 Inflation . . . 223

8.7 An electrical interpretation . . . . . 223
8.8 Experiments . . . . . 227
8.8.1 Discrimination efficiency . . . . . 228
8.8.2 Impact of inflation . . . . . 234
8.8.3 CPU time . . . . . 234
8.8.4 Approximating $\mathcal{K}$-walks with limited $\mathcal{K}$-walks . . . . . 236
8.8.5 Implementation issues . . . . . 238
8.9 Summary and discussion . . . . . 238

**Conclusion** **243**

**Bibliography** **249**

# Introduction

Sequential data are encountered in many contexts of everyday life and in numerous scientific applications. Think for instance to SMS typeset with mobile phones, web pages reached while crossing hyperlinks, parts of speech, musical pieces, hand-written digits, system logs, climatic informations or DNA samples, to name a few. All these examples are defined by the succession of events forming sequences. These events can be the characters entered while writing SMS, visited web pages or pronounced phonemes. Generating such sequences of events defines a *sequential process*. Mathematical models of sequential processes are useful for making predictions about future events or to shed light on how sequences are generated. Stepping back to our examples, mobile phones now integrate systems to facilitate message typesetting on numerical keyboards. Such systems are based on the prediction of words from their first characters. Besides, scientists can discover meaningful relations between observations, *e.g.* DNA sequences, and some unobservable phenomenons revealed by the model.

Traditionally, models are designed by experts of the application domains such as phoneticians for modeling speech or meteorologists for modeling the weather evolution. That is, domain experts try to include their knowledge inside the mathematical models. While this approach has successful applications, it has some severe limitations. First, it can be hard to synthesize a sufficient knowledge to build accurate models or sometimes such a knowledge is simply not *a priori* available. For instance, specifying rules for recognizing occurrences of words or groups of words in a speech signal is very complicated. Systems based on such tentative rules perform very poorly in speech recognition applications. Besides, models need sometimes to be adjusted after some task adaptation. For instance, a model for hand-written digit recognition should adapt to the end-user and should gradually improve its performance as more input data have been observed.

The *machine learning* approach attempts to automatically learn or induce models from empirical observations. That is, knowledge of the ap-

plication domain is acquired directly from data rather than from experts. Such an approach clearly circumvents the previously mentioned issues as the reliance to the experts is limited or avoided and models can be updated whenever more data become available. This thesis is mainly concerned with the general problem of modeling sequential processes from data. To do so, we consider probabilistic models called *Markov models*. Probabilistic models are mathematical models assigning probabilities to event realizations. The major strength of such models is their ability to cope with *uncertainty*. Indeed, data observed in real-life contexts are never crystal clear. For instance, given current climatic observations, the weather can evolve in many different ways. Hence, rather than predicting univocally the next events of a process, probabilistic models provides the probability for events to occur. Generating data according to some probabilistic model defines a *stochastic process*. Markov chains (MC), introduced by the Russian mathematician Andrei Markov in the beginning of the 20th century, are one of the most fundamental contributions to the field of stochastic processes. These models make the assumption that the next outcome of a process only depends on a fixed number of previous events. This assumption is not necessarily satisfied in real-life processes since outcomes are sometimes conditioned by the realization of some events arbitrarily far in the past. For instance, the weather on a given day may depend on climatic elements such as anticyclones formed at a much earlier time. Such phenomenons are called *long-term dependencies*.

Hidden Markov Models (HMMs) are an extension of MC introduced to model sequential processes including unobservable phenomenons. For instance, given a DNA sequence, one can observe the four nucleotides `A,C,T,G` but not biological phenomenons such as the process of protein creation. In HMMs, unobservable events are represented by *hidden states* whereas observations are modeled by an *emission process* depending on hidden states. In the early nineties, these models have become very popular in many pattern recognition applications. In the present work, we focus especially on the hidden aspect of HMMs to encode complex temporal dynamics which may include long-term dependencies. Indeed, these models circumvent some of the limitations of traditional MC as, outcomes can potentially depend on any previous event realizations. Markov models can be graphically represented as state charts where transitions between states are labeled with a probability. Traversing such a graphical model according to the transition probabilities defines a *random walk*.

Markov models can answer questions such as (i) *Which* event will occur a couple of steps later? (ii) *How many times* will a particular event occur? and (iii) *When* does an event occur for the first time given the current

situation? The last question is of particular interest in this thesis and is mathematically formalized under the notion of *first passage times dynamics* of a process. That is, we focus on *when* events occur rather than on *which* event will occur next. This kind of analysis has applications in many areas such as finance, insurance and queuing systems, to name a few.

## Problems and approach

This thesis attempts to solve the three following problems:

1. **HMM induction**: The general problem consists in automatically learning a HMM from a data sample. The induction of such models is twofold: (i) the model structure has to be defined and (ii) the probabilistic parameters of the model have to be estimated. While efficient techniques for parameter estimation have been provided, there is no widely adopted strategy for learning the structure of these models. Furthermore, models learned with the existing techniques generally fail to incorporate the long-term behavior of the target process from which the sample has been generated. Our specific objective is to learn the structure and the parameters of HMMs to accurately model the temporal dynamics, possibly including long-term dependencies, of the target process.

2. **Sequence classification**: The objective is to automatically classify sequences in a given set of classes or labels. For instance, one may want to automatically classify musical pieces according to their genre: classical, jazz, pop, ... or to find where a protein is located inside the cell from a set a possible locations: mitochondria, membrane, cytoplasm .... Hence, the prediction does not concern the future outcomes of a process but the class or the label of a given sequence. While the problem of classifying data represented as vectors has been studied for some decades, classification techniques dealing with structured data, such as sequences, have received more attention recently. Our objective is to provide new methods to this problem, relying on dynamical properties in sequences.

3. **Graph mining**: This problem is somewhat different from the two preceding ones as, the input data no longer consist of sequences but of a network, represented mathematically by a *graph*, made of interconnected entities. Such a network can for instance be a roadmap where the entities are locations and the connections are roads or highways between these locations. In this context, one is not necessary concerned

by the whole network but by some specific entities (also called nodes) of interest, *e.g.* places to travel or bioentities in a given metabolic pathway. The problem we want to solve is to extract a part of the original network, called a *subgraph*, that best models or explains the relationships between the entities of interest.

The general approach proposed to tackle these three problems is based on first passage times (FPT). Given two events $e_1$ and $e_2$ occurring in a process, the FPT between $e_1$ and $e_2$ are the times between a realization of $e_1$ and the next realization of $e_2$. Such features can be extracted from input sequences or computed from a model. We propose to learn HMM to fit the FPT observed in the learning sequences. Such a *fitting criterion* is relevant to specifically learn the temporal dynamics of the target process and the long-term dependencies it possibly contains. In the context of sequence classification, we assume that sequences labeled differently are drawn from stochastic processes having distinct FPT dynamics. That is, FPT are considered as a *discriminative information* used to classify sequences. Finally, the mining problem is approached by defining a dynamical view of the input graph consisting of a MC. Random walks can be performed on this MC and the FPT between nodes of interest during these walks are computed. These FPT are used to define a *relevance criterion* to extract subgraphs.

## Main contributions

The main contributions of the present work are the following:

- The POMMStruct algorithm for estimating the model structure and parameters of a *Partially Observable Markov Models* (POMMs) is introduced (see chapter 5). The induced model is estimated so as to fit the FPT dynamics observed in sequences. The structure is learned by iteratively adding states to the model and by trimming infrequently used transitions. The model parameters are estimated using the novel POMMPHit algorithm which maximizes the likelihood of the observed FPT.

- Two sequence classification approaches based on FPT between subsequences are proposed (see chapter 7). FPT are modeled with discrete phase-type distributions using a novel algorithm called PHit. The first approach builds a maximum *a posteriori* model from phase-type distributions while the second method uses a kernel relying on these distributions.

- A novel approach to mine a subgraph that best explains the relation between prescribed nodes of interests in a large graph is proposed (see chapter 8). The subgraph is extracted according to a new relevance criterion based on specific random walks called $\mathcal{K}$-walks. An efficient procedure for computing this criterion while considering walks of limited length is proposed, allowing one to deal with large graphs.

## Thesis roadmap

This thesis is organized in four parts: (I) Background, (II) Hidden Markov Model induction, (III) Sequence classification and (IV) Graph mining.

Part I provides the background material required to understand existing and novel approaches presented in this thesis. Chapter 1 reviews the theory of discrete time MC including basic definitions, the stationary distribution, absorbing MC techniques to analyze passage times, phase-type distributions and techniques for estimating MC from a data sample. Absorbing MC techniques are used throughout this thesis to compute FPT whenever needed. Next, chapter 2 presents an overview of HMMs containing basic definitions, links with probabilistic automata and three main problems identified by Rabiner [107] with their respective solutions. This material is a prerequisite to understand part II concerning HMM induction. Finally, chapter 3 provides an introduction to kernel methods including the basics about the statistical learning theory of Vapnik, kernel functions, Support Vector Machines (SVM), notions about convex optimization, relations between SVM and the statistical learning theory and implementation issues. This material is used in part III dealing with sequence classification problems.

Part II is concerned with the induction of HMMs. Chapter 4 presents an overview of existing techniques towards this objective. It includes methods learning the structure through parameter estimation, techniques based on state merging and state splitting, and discriminative induction approaches. Chapter 5 presents original material related to the HMM induction problem. New graphical models called Partially Observable Markov Models (POMMs) are introduced. We establish the equivalence between these models and HMMs. We also provide links between POMMs and MC. The FPT dynamics in POMMs is studied and connections between this dynamics and the model structure are underlined. The problem of modeling processes containing long-term dependencies is discussed. Next, we propose a novel induction algorithm, called POMMStruct, for estimating the structure and the probabilistic parameters of POMMs. The iterative state addition,

the transition trimming technique and the FPT fitting procedure are detailed. Finally, practical experiments are conducted with POMMStruct and other competing approaches on several sequence modeling tasks.

Part III deals with the sequence classification problem. Chapter 7 reviews several existing approaches to this problem. It includes generative classification techniques such as the Naive Bayes classifier and extensions based on variable-order MC. We also review discriminative techniques such as discriminatively trained MC and kernel methods. Chapter 7 presents a novel approach to the sequence classification problem. We propose a general scheme to perform classification based on FPT. A feature selection procedure to identify the most informative FPT features is presented. We introduce a new technique, called PHit, for fitting passage times with phase-type distributions. Two sequence classifiers based on the estimated PH distributions are proposed. Finally, the performance of these new classifiers is assessed with respect to state-of-the-art approaches.

Part IV is concerned with the problem of mining relevant subgraphs in large networks. Chapter 8 first presents the general context and provides intuition about random walks in such mining problems. Specific random walks called $\mathcal{K}$-walks and relevance criteria based of these walks are introduced. The computation of these relevance measures, using absorbing MC techniques is detailed. Alternatively, we propose to use $\mathcal{K}$-walks of a prescribed length or up to a prescribed length. An efficient algorithm for computing relevances for such walks is presented. Techniques for extracting subgraphs based on the proposed relevance measures are discussed. We also introduce an inflation procedure to improve the discrimination efficiency of the proposed techniques. An electrical interpretation of the $\mathcal{K}$-walk approach is given. Finally, experiments are conducted with the proposed approaches on a biological network and on artificially generated graphs.

The original contributions to the present thesis are contained in the following publications:

[21] J. Callut and P. Dupont. A Markovian approach to the induction of regular string distributions. In *Grammatical Inference: Algorithms and Applications*, number 3264 in Lecture Notes in Artificial Intelligence, pages 77–90, Athens, Greece, 2004. Springer Verlag

[23] J. Callut and P. Dupont. Inducing hidden markov models to model long-term dependencies. In *16th European Conference on Machine*

*Learning (ECML)*, number 3720 in Lecture Notes in Artificial Intelligence, pages 513–521, Porto, Portugal, 2005. Springer Verlag

[24] J. Callut and P. Dupont. Sequence discrimination using phase-type distributions. In *17th European Conference on Machine Learning (ECML)*, number 4212 in Lecture Notes in Artificial Intelligence, pages 78–89, Berlin, Germany, 2006. Springer Verlag

[43] P. Dupont, J. Callut, G. Dooms, J.-N. Monette, and Y. Deville. Relevant subgraph extraction from random walks in a graph. Technical Report RR 2006-07, INGI, UCL, 2006

[25] J. Callut and P. Dupont. Learning hidden markov models from first passage times. In *18th European Conference on Machine Learning (ECML)*, number 4701 in Lecture Notes in Artificial Intelligence, pages 91–103, Warsaw, Poland, 2007. Springer Verlag

# Part I

# Background

# Chapter 1

# Discrete time Markov chains

*This chapter reviews the basic theory about discrete time Markov chains (MC). These models are one of the most commonly used techniques to model sequential processes. Furthermore, MC can also model discrete time durations with the so-called phase-type distributions. These distributions are of special interest in this thesis since they are used to model the first passage times (FPT) dynamics in sequences. The FPT are used to induce Hidden Markov models (HMM) (see chapter 5), to perform discrete sequence classification (see chapter 7) and to mine relevant subgraphs in large networks (see chapter 8). The present chapter is organized as follows. Section 1.1 reviews the classical definition of a MC as well as the MC state set partitioning into equivalence classes. The Perron-Frobenius theorem applied to MC transition matrices is presented in section 1.2. Consequences of this theorem are important since it characterizes the long-term dependencies possibly embedded in a process (see section 5.4). In particular, HMMs can model long-term dependencies, provided an adequate topology is used. The proposed induction technique (see section 5.5) automatically learns such a structure by fitting the FPT observed in the learning sample. Section 1.3 presents two fundamental quantities characterizing the dynamics of random walks in MC, namely the stationary distribution and the FPT. Section 1.4 is concerned with the distribution of the FPT, i.e with discrete phase-type distributions. Finally, estimation and smoothing techniques to learn a MC from a data sample are presented in section 1.5. For a more detailed introduction to the MC theory, the reader is referred to the classical text books [73, 98]. The theory of non-negative and stochastic matrices is thoroughly studied in [117]. More complete references about phase-type distributions are [94, 80]. State-of-the-art estimation and smoothing techniques of MC are presented in [75].*

## 1.1 Definitions

The classical definition of a discrete time Markov chain is given hereafter.

**Definition 1 (Discrete Time Markov Chain)** *A discrete time* Markov Chain (MC) *is a stochastic process* $\{X_t \mid t \in \mathbb{N}\}$ *where the random variable* $X$ *takes its value at any discrete time* $t$ *in a countable set* $Q$ *and such that:*

$$P[X_t = q \mid X_{t-1}, X_{t-2}, \ldots, X_0] = P[X_t = q \mid X_{t-1}, \ldots, X_{t-p}]. \tag{1.1}$$

*This condition states that the probability of the next outcome only depends on the last* $p \in \mathbb{N}$ *values of the process (Markov property). When the set* $Q$ *is finite, the process forms a* $p$ *order finite state MC.*

If the outcome probabilities of the process are invariant to a time shift, the MC is said to be *homogeneous*.

**Definition 2** *An order* $p$ *MC is* homogeneous *if*

$$P[X_t = q \mid X_{t-1}, \ldots, X_{t-p}] = P[X_p = q \mid X_{p-1}, \ldots, X_0] \tag{1.2}$$

*holds for every* $t \geq p \geq 0$.

In the sequel, a MC refers to an order 1 homogeneous model unless stated otherwise. A MC can be represented by a 3-tuple $T = \langle Q, A, \boldsymbol{\iota} \rangle$ where $Q$ is a finite set of states, $A$ is a $|Q| \times |Q|$ transition probability matrix such that $A_{qq'} = P[X_t = q' \mid X_{t-1} = q]$ and $\boldsymbol{\iota}$ is a $|Q|-$dimensional vector representing the initial probability distribution, *i.e.* $\iota_q = P[X_0 = q]$. In a proper MC, each line of $A$ as well as the initial vector $\boldsymbol{\iota}$ must be stochastic (*i.e.* sum up to one). It is often convenient to represent a MC as a directed graph with a set of nodes $Q$ and having a weighted adjacency matrix $A$. To illustrate this, let us consider an intuitive example inspired from [73] which consists of modeling the weather evolution in the Europe capital. For the sake of simplicity, the weather can be nice (N), rainy (R) or snow (S). Initially the weather is assumed to be nice. The weather changes as follows: (i) if the weather is nice, the next day will be rainy, (ii)if it snowing or raining, there is an even chance that the weather remains the same on the next day and (iii) if the weather changes from snow or rain, there is an even chance to have a nice day. A MC $T_{\text{Weather}}$ can be formed by defining the state set $Q = \{N, R, S\}$, the initial vector $\boldsymbol{\iota}^T = (1, 0, 0)$ and the transition matrix :

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 1/4 & 1/2 & 1/4 \\ 1/4 & 1/4 & 1/2 \end{pmatrix} \tag{1.3}$$

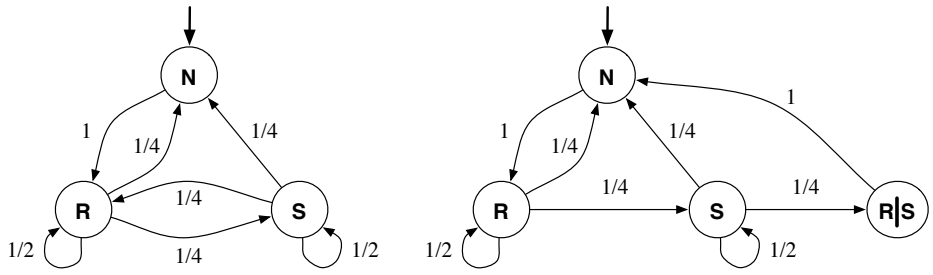



Figure 1.1: The evolution of the weather in the Europe capital. Left: the weather evolution modeled by an order 1 MC. Right: the weather evolution modeled by an order 2 MC.

The representation of this chain as a graph is depicted in the left part of Figure 1.1.

Let us note that it is also possible to represent a higher order MC by an order 1 MC $T = \langle Q, A, \boldsymbol{\iota} \rangle$. For instance, let us modify our weather example in the following way. When the weather changes from a rainy day, two case are considered: if the day before the rain was a snowy day then the next day will be nice, otherwise the weather evolves as described above. Clearly, this behavior cannot be modeled by a MC of order less than 2 with the same state set $Q = \{N, R, S\}$. This order 2 MC can however be encoded by an order 1 MC with an extended state set $Q = \{N, R, S, R|S\}$ where the state $R|S$ stands for a rainy day following snow. The initial vector is $\boldsymbol{\iota}^T = (1, 0, 0, 0)$ and the transition matrix is defined as

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1/4 & 1/2 & 1/4 & 0 \\ 1/4 & 0 & 1/2 & 1/4 \\ 1 & 0 & 0 & 0 \end{pmatrix} \tag{1.4}$$

The entry $A_{R|S,N}$ encodes the order 2 conditional probability $P[X_t = N | X_{t-1} = R, X_{t-2} = S]$. This MC is shown in the right part of Figure 1.1.

Once a MC $T = \langle Q, A, \boldsymbol{\iota} \rangle$ is defined, a process instance is called a *random walk*.

**Definition 3 (Random walk)** *Given a MC, $T = \langle Q, A, \boldsymbol{\iota} \rangle$, a* random walk *is defined on $T$ as follows. The process starts on a state $q_0$ according to the initial distribution $\boldsymbol{\iota}$. The process next moves to some state $q$ according to the transition matrix $A$. Repeating this operation $n \in \mathbb{N}$ times results in a $n$-steps* random walk.

It can be interesting to know where the process will be $k$ steps after initialization. For instance, in the weather example it could be used to predict the weather a couple of days later. This question can be answered by using the powers of the transition matrix $A$. Indeed, the entry $(A^k)_{q_0 q}$ contains the probability to go from state $q_0$ to state $q$ in exactly $k$ steps, i.e $P[X_k = q \mid X_0 = q_0]$. Therefore, the probability that the process reaches state $q$ after $k$ steps is simply computed as

$$\sum_{q_0 \in Q} P[X_0 = q_0] P[X_k = q \mid X_0 = q_0] = (\boldsymbol{\iota}^T A^k)_q \tag{1.5}$$

where $(\boldsymbol{\iota}^T A^k)_q$ is the $q$-th entry of the row vector $\boldsymbol{\iota}^T A^k$. Let us note that $\langle Q, A^k, \boldsymbol{\iota} \rangle$ is also a proper MC for all $k \in \mathbb{N}$.

The states of a MC can be classified into two categories : *transient states* and *recurrent states*. In the original Markov chain theory (see for instance [117]), transient and recurrent states are respectively referred to as *inessential* and *essential states*.

**Definition 4** *Given a MC $T = \langle Q, A, \boldsymbol{\iota} \rangle$, a state $q \in Q$ is a* recurrent *state if*

$$P[\inf_{t>0}\{t \mid X_t = q\} < \infty \mid X_0 = q] = 1 \tag{1.6}$$

That is, once a recurrent state $q$ is reached by the process, there is a probability one to come back in $q$ in a finite time. Equivalently, if $q$ is reached by the process, the average number of times it is visited is unbounded.

**Definition 5** *Given a MC $T = \langle Q, A, \boldsymbol{\iota} \rangle$, a state $q \in Q$ is a* transient *state if*

$$P[\inf_{t>0}\{t \mid X_t = q\} < \infty \mid X_0 = q] < 1 \tag{1.7}$$

When the process reaches a transient state $q$, there is a strictly positive probability that the process will never go back to $q$ again. Consequently, the average number of times a transient state is reached is always bounded.

Let us now introduce the "*leads stochastically*" relationship between states partitions the MC state set $Q$.

**Definition 6** *Given a MC $T = \langle Q, A, \boldsymbol{\iota} \rangle$, a state $q \in Q$ leads stochastically ($\rightsquigarrow$) to a state $q' \in Q$, if there exists a number $k \in \mathbb{N}$ such that $(A^k)_{qq'} > 0$.*

The state set $Q$ can now be partitioned in classes according to an equivalence relationship between two states defined as $q \equiv q' \iff q \rightsquigarrow q' \wedge q' \rightsquigarrow q$. One can distinguish two kinds of equivalence classes: *transient classes* and *final classes*.

**Definition 7** *Given a MC* $T = \langle Q, A, \boldsymbol{\iota} \rangle$*, an equivalence class* $\mathcal{C} \subsetneq Q$ *is* transient *if*

$$\forall q \in \mathcal{C}, \exists q' \in Q \setminus \mathcal{C} : q \rightsquigarrow q' \wedge q' \not\rightsquigarrow q \tag{1.8}$$

That is, the states of a transient class $\mathcal{C}$ lead to at least one state $q'$ outside the class but not conversely. The exterior state(s) $q'$ can belong either to a transient class or to a final class.

**Definition 8** *Given a MC* $T = \langle Q, A, \boldsymbol{\iota} \rangle$*, an equivalence class* $\mathcal{C} \subseteq Q$ *is* final *if*

$$\forall q \in \mathcal{C}, \forall q' \in \mathcal{C}' \neq \mathcal{C} : q \not\rightsquigarrow q' \tag{1.9}$$

*where* $\mathcal{C}'$ *is another equivalence class defined on* $Q$*.*

That is, once a final class is reached the process will remain in this class forever with probability 1. It can be shown that a transient class only contains transient states and that a final class only contains recurrent states. In addition, a MC always contains at least one final class while it might contain no transient class. According to this class partitioning, several kinds of MC can be distinguished. A particular case of interest is when the chain only contains one self-communicating class, *i.e.*, when its transition graph is strongly connected.

**Definition 9** *A MC* $T = \langle Q, A, \boldsymbol{\iota} \rangle$ *is* irreducible *if* $T$ *is made of a single final class.*

In some references (for instance in the text book [73]), an irreducible chain is called *ergodic*. Irreducible MC can be further refined by introducing the notion of *(a)periodic states*.

**Definition 10** *Given a MC* $T = \langle Q, A, \boldsymbol{\iota} \rangle$*, the* period *of a state* $q \in Q$ *is defined as*

$$period(q) = \begin{cases} 0 \quad \text{if } \forall\, t \in \mathbb{N}^0,\ P[X_t = q \mid X_0 = q] = 0, \\ GCD(\{t \in \mathbb{N}^0 \mid P[X_t = q \mid X_0 = q] > 0\}) \text{ otherwise,} \end{cases} \tag{1.10}$$

*where* $GCD(.)$ *denotes the greatest common divisor of a set of integers.*
*If* $period(q) > 1$ *then* $q$ *is a* periodic state *otherwise* $q$ *is an* aperiodic state.

That is, a state $q$ has a period $p > 1$ if each walk from $q$ to itself has a length which is a multiple of $p$. Let us note that every states in an equivalence class have the same period. Hence, the period of a class is defined as the period of every states belonging to this class. Let us note, that, if a state $q$ in a class $\mathcal{C}$ has a self loop, *i.e.* $A_{qq} > 0$, then $\mathcal{C}$ is necessarily aperiodic. According to its periodicity, an irreducible MC can be either *regular* or *cyclic*.

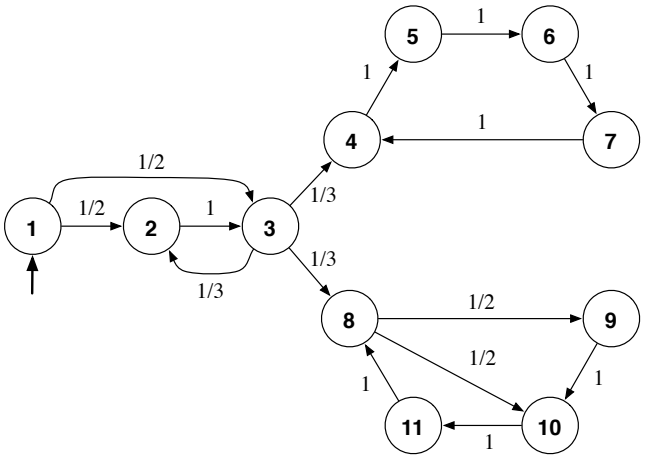



Figure 1.2: A Markov chain containing two transient classes and two final classes.

**Definition 11** *An irreducible MC is* regular *if all its states are aperiodic.*

A necessary and sufficient condition for a MC $T = \langle Q, A, \boldsymbol{\iota} \rangle$ to be regular is that there exist a number $k \in \mathbb{N}$ such that for all $k' \geq k$, all entries of $A^{k'}$ are strictly positive. A matrix with non-negative entries having this property is called a *primitive* matrix.

**Definition 12** *An irreducible MC is* cyclic *if all its states are periodic.*

To illustrate these notions, let us consider the example depicted in Figure 1.2. This MC contains two transient classes $\mathcal{C}_1 = \{1\}$ and $\mathcal{C}_2 = \{2, 3\}$ as well as two final classes $\mathcal{C}_3 = \{4, 5, 6, 7\}$ and $\mathcal{C}_4 = \{8, 9, 10, 11\}$. The final class $\mathcal{C}_3$ is periodic since all its states have a period equal to 4. Let us note that the subchain induced by the states of $\mathcal{C}_3$ with a proper initial distribution forms a cyclic MC. The final class $\mathcal{C}_4$ is aperiodic since it is possible to go from state 8 to itself in 3 or 4 steps and that $GCD(3, 4) = 1$. Note also that the subchain induced by the states of $\mathcal{C}_4$ with a proper initial distribution forms a regular MC.

Another particular class of MC is called the *absorbing* MC.

**Definition 13** *A MC is* absorbing *if all its final classes contain a single state.*

In an absorbing MC, a recurrent state $q$ is called an *absorbing state* and $A_{qq} = 1$. In such a model, the state set $Q$ can be partitioned into the

absorbing state set $Q_A$ and its complementary set, the transient state set $Q_T$. The initial distribution and the transition matrix of an absorbing MC can be represented in canonical block form by setting the indices of the absorbing states at the end :

$$\boldsymbol{\iota}^T = (\boldsymbol{u}^T \quad \mathbf{0}^T), \qquad A = \begin{pmatrix} F & E \\ O & I \end{pmatrix} \tag{1.11}$$

where $\boldsymbol{u}$ is a $|Q_T|$-dimensional vector, $\mathbf{0}$ is a $|Q_A|$-dimensional vector of zeros, $F$ is a $|Q_T| \times |Q_T|$ transition submatrix between transient states, $E$ is a $|Q_T| \times |Q_A|$ transition submatrix from transient states to absorbing states, $O$ is a $|Q_A| \times |Q_T|$ matrix of zeros and $I$ is a $|Q_A| \times |Q_A|$ identity matrix. Let us note that the matrix $F$ is substochastic, *i.e.* it contains at least one row the sum of which is less than 1. Furthermore, according to the structure of $\boldsymbol{\iota}$ it is assumed that the process always starts in a transient state, hence $\boldsymbol{u}$ is stochastic. A particular case is when the absorbing MC contains only a single absorbing state.

**Definition 14** *An absorbing MC is* reduced *if it contains a single absorbing state.*

## 1.2 The Perron-Frobenius theorem

The MC theory is closely related to linear algebra and especially to the theory of non-negative and stochastic matrices. The properties of such matrices characterize interesting behaviors of the process. For instance, the spectral values of the MC transition matrix directly provides information about the long-term dependencies possibly embedded in the process (see section 5.4). We quote here perhaps the most important result from the theory of non-negative matrices which is the Perron-Frobenius theorem [117] applied here to stochastic matrices. This theorem is also useful to prove other results such as the existence of a stationary distribution for regular MC (see section 1.3).

**Theorem 1 (Perron-Frobenius applied to stochastic matrices [117])** *Let $A$ be a $n_s \times n_s$ primitive matrix. Then, there exists an eigenvalue $\lambda_{max} = 1$ such that*

1. *strictly positive left and right eigenvectors are associated with $\lambda_{max}$,*
2. *$\lambda_{max} > |\lambda|$ for any eigenvalue $\lambda \neq \lambda_{max}$,*
3. *the eigenvectors associated with $\lambda_{max}$ are unique up to a multiplicative constant,*

4. *if a $n_s \times n_s$ matrix $A'$ is such that $0 \leq A'_{ij} \leq A_{ij}$ and that $\lambda'$ is an eigenvalue of $A'$ then $|\lambda'| \leq \lambda_{max}$. Moreover $|\lambda'| = \lambda_{max}$ implies $A' = A$,*

5. *$\lambda_{max}$ is a simple root of the characteristic equation of $A$.*

**Proof.** *see p. 4–7 in [117].*

## 1.3 Dynamics of random walks

This section presents fundamental quantities characterizing the dynamics of random walks in MC, namely the stationary distribution, the absorption times and absorption probabilities, and the First Passage Times (FPT).

### Stationary distribution

One can be interested in the long run behavior of the process, *i.e.* in the limiting probabilities

$$\lim_{t\to\infty} P[X_t = q] = \lim_{t\to\infty} (\boldsymbol{\iota}^T A^t)_q, \quad \forall q \in Q \tag{1.12}$$

Clearly the process will be in a final class. We concentrate here our attention on irreducible chains. For regular MC, these limiting probabilities are obtained by computing the *stationary distribution* of the chain.

**Theorem 2 (The stationary distribution for regular MC)** *Given a regular MC $T = \langle Q, A, \boldsymbol{\iota}\rangle$, the limiting probabilities $\lim_{t\to\infty} P[X_t]$ are obtained by computing the* stationary distribution *vector $\boldsymbol{\pi}$ such that $\boldsymbol{\pi}^T A = \boldsymbol{\pi}^T$.*

**Proof.**
*The Perron-Frobenius theorem implies that the primitive matrix $A$ has a unique largest eigenvalue $\lambda_{max} = 1$. Consequently, the sequence of matrices $A^t$ converges to a rank one matrix: $\lim_{t\to\infty} A^t = \boldsymbol{r}\boldsymbol{\pi}^T$ where $\boldsymbol{r}$ and $\boldsymbol{\pi}$ are respectively the right and the left eigenvectors of $A$ associated to $\lambda_{max}$. Moreover, since $A$ is stochastic, the right eigenvector $\boldsymbol{r} = \mathbf{1}$. Therefore, the limiting probabilities are given by*

$$\lim_{t\to\infty} \boldsymbol{\iota}^T A^t = \boldsymbol{\iota}^T \mathbf{1}\boldsymbol{\pi}^T = \boldsymbol{\pi}^T \tag{1.13}$$

■

The vector $\boldsymbol{\pi}$ is also known as the *equilibrium vector* or as the *steady-state vector*. This vector can be obtained by computing the left eigenvector of $A$ associated to the eigenvalue 1 and by normalizing it such that it sums up

to 1. The $q$-th entry of the vector $\boldsymbol{\pi}$ is equal to $\lim_{t\to\infty} P[X_t = q]$ and can also be interpreted as the expected proportion of the time the process in steady-state reaches state $q$. Let us note that the stationary distribution is completely independent of the initial distribution $\boldsymbol{\iota}$. A regular MC is started in steady-state when the initial distribution $\boldsymbol{\iota}$ is set to the stationary distribution $\boldsymbol{\pi}$.

For cyclic chains, limiting probabilities cannot be defined since states are only reached at specific time indices. However, even if the sequence $\{A^t\}_t$ diverges, a limiting average of the sequence, called the Euler sum, is computable (for details see Theorem 5.1.1 in [73]). This yields the stationary distribution vector $\boldsymbol{\pi}$ which is computed exactly as the stationary vector of a regular chain, *i.e.* by solving $\boldsymbol{\pi}^T A = \boldsymbol{\pi}^T$ (for details see Theorem 5.1.2 in [73]). It can also be interpreted as the proportion of time the process reaches each state in the long run.

To illustrate this concept, let us compute the limiting probabilities for the order 1 MC modeling the weather in the Europe capital (left side of Figure 1.1). This chain is clearly regular as it contains a single final class which is aperiodic (for instance, states $R$ and $S$ have a self-loop). The stationary distribution is $\boldsymbol{\pi} = (N : 0.2, R : 0.53, S : 0.27)$ which can be interpreted as follows: 20% of the time the weather is nice, 53 % of the time it is rainy and 27% of the time there is snow.

### Absorbing MC techniques

This section presents techniques to analyze the process behavior before reaching absorbing states. The scope of these techniques goes beyond absorbing MC. They can be used to study the process behavior before reaching a particular state $q$ which needs not to be absorbing. Such methods are also used to define the FPT distributions in Partially Observable Markov Models (see section 5.3). First, let us introduce the notion of *passage time* during random walks on a MC $T$:

**Definition 15 (Passage Time)** *Given a MC, $T = \langle Q, A, \boldsymbol{\iota}\rangle$, the* passage time *is a function* $\mathrm{pt} : Q \times Q \to \mathbb{N}$ *such that* $\mathrm{pt}(q_0, q)$ *is the number of times the process reaches the state $q$, leaving initially from state $q_0$ :*

$$\mathrm{pt}(q_0, q) = |\{t \in \mathbb{N} \mid X_t = q, X_0 = q_0\}| \tag{1.14}$$

*The Mean Passage Time denotes the expectation of this quantity:*

$$\mathrm{mpt}(q_0, q) = \mathbb{E}\left[\mathrm{pt}(q_0, q)\right] \tag{1.15}$$

That is, $\mathrm{pt}(q_0, q)$ is the number of times the process reaches state $q$ during random walks started in state $q_0$. In the sequel, it is assumed that every states of the chain are reachable by the process, *i.e.* $\forall q \in Q, \exists\ t \in \mathbb{N}, (\boldsymbol{\iota}^T A^t)_q > 0$. The mean passage time in an absorbing state $q$ is clearly unbounded. For transient states, the mean passage time is provided by the following theorem.

**Theorem 3 (The fundamental matrix of an absorbing MC)** *Given an absorbing MC $T = \langle Q, A, \boldsymbol{\iota} \rangle$ in canonical form and two transient states $q_0, q \in Q_T$, the mean passage time* $\mathrm{mpt}(q_0, q)$ *can be computed as*

$$\mathrm{mpt}(q_0, q) = (I - F)^{-1}_{q_0 q} \tag{1.16}$$

**Proof.**
*The passage time in a transient state $q$ can be decomposed as a sum of indicator variables*[1]*:* $\mathrm{pt}(q_0, q) = \sum_{t=0}^{\infty} \mathbb{1}\{X_t = q \mid X_0 = q_0\}$*. Therefore, the expectation of* $\mathrm{pt}(q_0, q)$ *is obtained as follows:*

$$\begin{aligned} \mathrm{mpt}(q_0, q) &= \mathbb{E}\left[\sum_{t=0}^{\infty} \mathbb{1}\{X_t = q \mid X_0 = q_0\}\right] = \sum_{t=0}^{\infty} \mathbb{E}\left[\mathbb{1}\{X_t = q \mid X_0 = q_0\}\right] \\ &= \sum_{t=0}^{\infty} P[X_t = q \mid X_0 = q_0] = \sum_{t=0}^{\infty} (F^t)_{q_0 q} = \left(\sum_{t=0}^{\infty} F^t\right)_{q_0 q} \end{aligned} \tag{1.17}$$

*We shall show that $\sum_{t=0}^{\infty} F^t$ actually converges to $(I - F)^{-1}$. First, let us note that*

$$(I - F)(I + F + \ldots + F^{t-1}) = I - F^t, \quad \forall t \geq 1 \tag{1.18}$$

*Moreover, the sequence of matrices $F^t$ converges to the zero matrix as $F$ is substochastic (for details, see [117] p. 120). For a sufficiently large number $t \in \mathbb{N}$, the matrix $(I - F)$ is non-singular since $det(I - F)det(I + F + \ldots + F^{t-1}) = det(I - F^t) \neq 0$. Letting $t$ tend to infinity yields:*

$$\begin{aligned} \lim_{t\to\infty}(I - F)(I + F + \ldots + F^{t-1}) &= I \\ \iff \lim_{t\to\infty}(I + F + \ldots + F^{t-1}) &= (I - F)^{-1} \end{aligned} \tag{1.19}$$

■

The matrix $(I-F)^{-1}$ plays an important role in the MC theory and is called the *fundamental matrix* of an absorbing MC. The mean passage times in an absorbing MC are equivalently computed by the following recurrence:

$$\mathrm{mpt}(q_0, q) = \delta_{q_0,q} + \sum_{q' \in Q_T} F_{q'q} \mathrm{mpt}(q_0, q') \tag{1.20}$$

[1] The value of an indicator variable $\mathbb{1}\{X = x\}$ is 1 when the random variable $X$ equals $x$ and 0 otherwise. The expectation of an indicator variable is simply $\mathbb{E}\left[\mathbb{1}\{X = x\}\right] = P[X = x]$.

where $\delta_{q_0,q} = 1$ if $q = q_0$ and 0 otherwise. The mean passage time in a state $q$ is the sum of the mean passage times over each transient state $q'$ weighted by the transition probability $F_{q'q}$. Furthermore, if $q$ is the starting state, a contribution of 1 must be added to take the initial position into account. If $N$ denotes a $|Q_T| \times |Q_T|$ matrix such that $N_{q_0 q} = \text{mpt}(q_0, q)$, the matrix form of equation (1.20) is given by

$$\begin{aligned} N &= I + FN \\ \iff N &= (I - F)^{-1} \end{aligned} \tag{1.21}$$

which, again, shows that the mean passage times are obtained by computing the fundamental matrix. When the MC is started according to the transient initial distribution $\boldsymbol{u}$, the mean passage times for all transient states are obtained by computing the vector $\boldsymbol{u}^T(I - F)^{-1}$. A closely related notion is the *time to absorption*.

**Definition 16 (Time to absorption)** *Given an absorbing MC* $T = \langle Q, A, \boldsymbol{\iota} \rangle$, *the* time to absorption *is a function* $\text{ta} : Q_T \to \mathbb{N}$ *such that* $\text{ta}(q_0)$ *is the time taken by the process to reach an absorbing state when it starts in a transient state* $q_0$*:*

$$\text{ta}(q_0) = \inf_t \{t \in \mathbb{N}^0 \mid X_t \in Q_A, X_0 = q_0\} \tag{1.22}$$

*The mean time to absorption is defined as the expectation of this quantity:*

$$\text{mta}(q_0) = \mathbb{E}\left[\text{ta}(q_0)\right] \tag{1.23}$$

The time to absorption can be thought of as the number of times the process reaches transient states before absorption. Therefore the mean time to absorption can be decomposed as the sum of the mean passage times over all transient states :

$$\text{mta}(q_0) = \sum_{q \in Q_T} \text{mpt}(q_0, q) = ((I - F)^{-1}\mathbf{1})_{q_0} \tag{1.24}$$

The mean time to absorption is equivalently computed by the following recurrence:

$$\text{mta}(q_0) = 1 + \sum_{q \in Q_T} F_{q_0 q} \text{mta}(q) \tag{1.25}$$

If $\boldsymbol{\tau}$ denotes a $|Q_T|$-dimensional vector such that $\tau_q = \text{mta}(q)$, the matrix form of equation (1.25) is given by

$$\begin{aligned} \boldsymbol{\tau} &= \mathbf{1} + F\,\boldsymbol{\tau} \\ \iff \boldsymbol{\tau} &= (I - F)^{-1}\mathbf{1} \end{aligned} \tag{1.26}$$

When the MC is started according to the transient initial distribution $\boldsymbol{u}$, the mean time to absorption is given by $\boldsymbol{u}^T(I-F)^{-1}\mathbf{1}$. An interesting related notion called the *absorption probability* can also be obtained from the fundamental matrix.

**Definition 17 (Absorption probability)** *Given an absorbing MC $T = \langle Q, A, \boldsymbol{\iota}\rangle$, the* absorption probability *of an absorbing state $q \in Q_A$ is the probability that the process get absorbed into state $q$ while starting from state $q_0 \in Q_T$:*

$$P_{trap}[q \mid X_0 = q_0] = P[\inf_t\{X_t = q\} < \infty \mid X_0 = q_0] \tag{1.27}$$

This probability can be computed as follows:

$$\begin{aligned} P_{trap}[q \mid X_0 = q_0] &= \textstyle\sum_{q' \in Q_T} \sum_{t=0}^{\infty} (F^t)_{q_0 q'} E_{q' q} \\ &= \textstyle\sum_{q' \in Q_T} (I-F)^{-1}_{q_0 q'} E_{q' q} \\ &= [(I-F)^{-1}E]_{q_0 q} \end{aligned} \tag{1.28}$$

If the MC contains only one absorbing state $q$ then $E = (I-F)\mathbf{1}$ and the absorption probability is one from any transient state. The absorption probability in any absorbing state $q$ when the MC is started according to the initial distribution $\boldsymbol{u}$ is given by $(\boldsymbol{u}^T(I-F)^{-1}E)_q$.

The absorbing MC techniques can be extended to the case of a general MC $T$ with final classes possibly containing several recurrent states. To do so, one has first to transform each final class $\mathcal{C}_i$ into a single absorbing state $q_{\mathcal{C}_i}$. This can be done without loss of generality since we study the process behavior before reaching a final class. Figure 1.3 shows such a transformation for the MC introduced in Figure 1.2. If $A$ denotes the transition matrix of the original MC having $r$ final classes $\mathcal{C}_1, \ldots, \mathcal{C}_r$, the transition matrix $A'$ of the transformed MC is obtained as follows (not explicitly specified elements are null):

$$A = \begin{pmatrix} F & E_{\mathcal{C}_1} & \cdots & E_{\mathcal{C}_r} \\ & A_{\mathcal{C}_1} & & \\ & & \ddots & \\ & & & A_{\mathcal{C}_r} \end{pmatrix} \quad \rightarrow \quad A' = \begin{pmatrix} F & \boldsymbol{e}_{\mathcal{C}_1} & \cdots & \boldsymbol{e}_{\mathcal{C}_r} \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{pmatrix} \tag{1.29}$$

where $F$ is a $|Q_T| \times |Q_T|$ transition submatrix between transient states, $E_{\mathcal{C}_i}$ is a $|Q_T| \times |\mathcal{C}_i|$ transition submatrix from transient states to the states in $\mathcal{C}_i$, $A_{\mathcal{C}_i}$ is a $|\mathcal{C}_i| \times |\mathcal{C}_i|$ transition submatrix between states in $\mathcal{C}_i$ and $\boldsymbol{e}_{\mathcal{C}_i} = E_{\mathcal{C}_i}\mathbf{1}$. The matrix $(I-F)^{-1}$ computes the mean passage times in transient states before

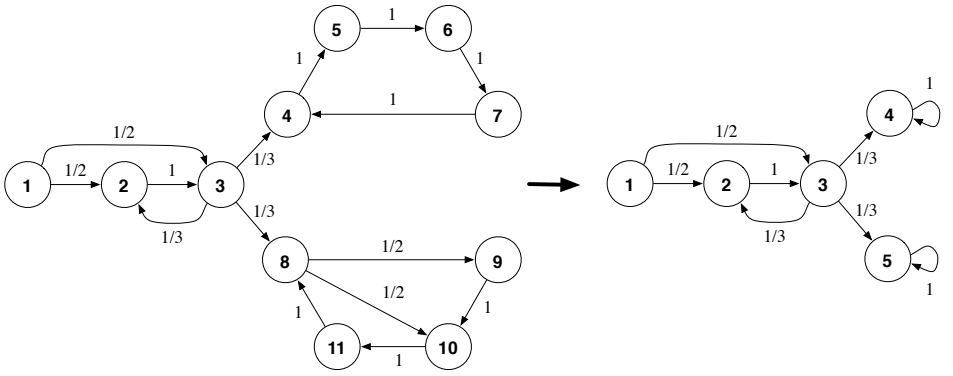



Figure 1.3: Transformation of the final classes of a MC into absorbing states.

the process get trapped into a final class. For recurrent states, the mean passage time is clearly unbounded. Similarly, the vector $\boldsymbol{\tau} = (I - F)^{-1}\mathbf{1}$ computes the expected time before the process reaches a final class. Finally, the probability to reach the final class $\mathcal{C}_i$ starting from state $q_0 \in Q_T$ is given by

$$P_{trap}[\mathcal{C}_i \mid X_0 = q_0] = [(I - F)^{-1}\boldsymbol{e}_{\mathcal{C}_i}]_{q_0} \tag{1.30}$$

One can also use absorbing MC techniques to study the process behavior before reaching a particular final class $\mathcal{C}_k$. Final classes are transformed into absorbing states as in equations (1.29). The absorption probabilities are provided by:

$$P_{trap}[q_{\mathcal{C}_k} \mid X_0 = q_0] = \begin{cases} [(I - F)^{-1}\boldsymbol{e}_{\mathcal{C}_k}]_{q_0} & \text{if } q_0 \in Q_T \\ 1 & \text{if } q_0 = q_{\mathcal{C}_k} \\ 0 & \text{otherwise} \end{cases} \tag{1.31}$$

Let us now define a reduced absorbing MC $T' = \langle Q_T \cup q_{\mathcal{C}_k}, A', \iota' \rangle$ modeling the original process behavior before reaching $\mathcal{C}_k$. The parameters of this model are defined in order to allow only random walks ending in $\mathcal{C}_k$. The initial probabilities for transient states $q \in Q_T$ are provided by

$$\begin{aligned} u'_{q_0} &= P[X_0 = q_0 \mid \inf_t\{X_t = q_{\mathcal{C}_k}\} < \infty] \\ &= \frac{P[\inf_t\{X_t = q_{\mathcal{C}_k}\} < \infty \mid X_0 = q_0]P[X_0 = q_0]}{P[\inf_t\{X_t = q_{\mathcal{C}_k}\} < \infty]} \\ &= \frac{P_{trap}[q_{\mathcal{C}_k} \mid X_0 = q_0]u_{q_0}}{\sum_{q_0 \in Q_T} P_{trap}[q_{\mathcal{C}_k} \mid X_0 = q_0]u_{q_0}} \end{aligned} \tag{1.32}$$

The transition probabilities for all $q, q' \in Q$ are given by:

$$\begin{aligned} A'_{qq'} &= P[X_{t+1} = q' \mid X_t = q, \inf_t\{X_t = q_{\mathcal{C}_k}\} < \infty] \\ &= \frac{P_{trap}[q_{\mathcal{C}_k} \mid X_{t+1} = q', X_t = q] P[X_{t+1} = q' \mid X_t = q]}{P_{trap}[q_{\mathcal{C}_k} \mid X_t = q]} \\ &= \frac{P_{trap}[q_{\mathcal{C}_k} \mid X_0 = q'] A_{qq'}}{P_{trap}[q_{\mathcal{C}_k} \mid X_0 = q]} \end{aligned} \tag{1.33}$$

Equivalently, the initial distribution $\boldsymbol{\iota}'$ and the transition matrix $A'$ in normal forms are readily obtained by defining a $|Q_T| \times |Q_T|$ diagonal matrix $D$ such that $D_q = P_{trap}[q_{\mathcal{C}_k} \mid X_0 = q]$ for all $q \in Q_T$:

$$\boldsymbol{\iota}' = (\boldsymbol{u}^T D / [\boldsymbol{u}^T D \mathbf{1}], 0)^T, \quad A' = \begin{pmatrix} D^{-1} F D & D^{-1} \boldsymbol{e}_{\mathcal{C}_k} \\ 0 & 1 \end{pmatrix} \tag{1.34}$$

Similarly, the fundamental matrix of the new absorbing MC is obtained from the fundamental matrix of the original MC as $(I - F')^{-1} = D^{-1}(I - F)^{-1} D$.

### First Passage Times

The first passage times are a generalization of the time to absorption since the destination state needs not to be absorbing.

**Definition 18 (First Passage Time)** *Given a MC $T = \langle Q, A, \boldsymbol{\iota} \rangle$, the* first passage time *(FPT) is a function* $\mathrm{fpt} : Q \times Q \to \mathbb{N}$ *such that* $\mathrm{fpt}(q_0, q)$ *is the number of steps before reaching state $q$ for the first time, leaving initially from state $q_0$.*

$$\mathrm{fpt}(q_0, q) = \inf_t \{t \in \mathbb{N}^0 \mid X_t = q \text{ and } X_0 = q_0\} \tag{1.35}$$

*The* Mean First Passage Time (MFPT) *denotes the expectation of this quantity:*

$$\mathrm{mfpt}(q_0, q) = \mathbb{E}\left[\mathrm{fpt}(q_0, q)\right] \tag{1.36}$$

Let $q_0$ and $q$ denotes two states belonging respectively to the equivalence classes $\mathcal{C}_0$ and $\mathcal{C}$. If the process, started in $q_0$, has a strictly positive probability to get trapped into a final class $\mathcal{C}_F \neq \mathcal{C}$ before reaching state $q$ then $\mathrm{mfpt}(q_0, q)$ is unbounded. Otherwise, one can simply compute $\mathrm{mfpt}(q_0, q)$ by transforming $q$ to be absorbing and by computing the mean time to absorption $\mathrm{mta}(q_0)$ as presented above. For a regular MC, the MFPT values can also be obtained by solving the following linear system:

$$\forall q_0, q \in Q, \mathrm{mfpt}(q_0, q) = \begin{cases} 1 + \sum_{q' \neq q} A_{q_0 q'} \mathrm{mfpt}(q', q) & \text{if } q_0 \neq q, \\ \dfrac{1}{\pi_q} & \text{otherwise.} \end{cases} \tag{1.37}$$

The values $\text{mfpt}(q,q)$ are usually called the *recurrence times*[2]. Interestingly, there is a close connection between the stationary distribution and the recurrence times in a regular MC. While $\pi_q$ can be interpreted as the proportion of the time the process in steady state reaches state $q$, the inverse of this quantity provides the mean time required to go from $q$ to itself. In the sequel, the matrix $M^{\text{FPT}}$ denotes a $|Q| \times |Q|$ matrix such that $M^{\text{FPT}}_{q_0 q} = \text{mfpt}(q_0, q)$. Importantly, the MFPT completely determine a regular MC in steady-state. That is, knowing the MFPT for any pair of states allows one to compute the transition matrix of the chain (and consequently its stationary distribution).

**Theorem 4** *Given a $|Q| \times |Q|$ matrix $M^{\text{FPT}}$ representing the MFPT for any pair of states $q, q' \in Q$ of a regular MC $T$, the transition probability matrix $A$ of $T$ can be obtained as follows:*

$$A = I + (\text{Diag}(M^{\text{FPT}}) - \mathbf{1}\mathbf{1}^T)(M^{\text{FPT}} - \text{Diag}(M^{\text{FPT}}))^{-1} \tag{1.38}$$

*where $\text{Diag}(M^{\text{FPT}})$ is a $|Q| \times |Q|$ diagonal matrix such that $\text{Diag}(M^{\text{FPT}})_q = M^{\text{FPT}}_{qq}$ for all $q \in Q$.* **Proof.** *see p.81 of [73].*

## 1.4 Discrete Phase-type distributions

In the previous section, the notion of time to absorption in an absorbing MC was presented. It was shown that the mean time to absorption can be obtained by computing the fundamental matrix. Furthermore, this technique can also be used to compute the MFPT in a general MC. The distribution of the times to absorption, or equivalently of the FPT in a general MC, is called a *distribution of phase-type*.

**Definition 19 (Discrete Phase-type (PH) Distribution)** *A probability distribution $\varphi(.)$ on $\mathbb{N}^0$ is a* distribution of phase-type *(PH) if and only if it is the distribution of the time to absorption in an absorbing MC.*

It should be pointed out that it is always possible to transform an absorbing MC with several absorbing states into a reduced one with the same distribution of the time to absorption. Hence, without loss of generality, we will only consider reduced MC in normal form, the last state being absorbing :

$$\boldsymbol{\iota}^T = (\boldsymbol{u}^T \quad 0), \quad A = \begin{pmatrix} F & \boldsymbol{e} \\ \mathbf{0} & 1 \end{pmatrix} \tag{1.39}$$

where $\boldsymbol{u}$ is a $|Q_T|$-dimensional *initial vector* for transient states, $F$ is an $|Q_T| \times |Q_T|$ matrix called the *phase generator*, $\boldsymbol{e}$ is an $|Q_T| \times 1$ vector called

[2] An alternative definition, $\text{mfpt}(q,q) = 0$, is possible when it is not required to leave the initial state before reaching the destination state for the first time [98].

the *absorption vector* and $\mathbf{0}$ is an $1 \times |Q_T|$ vector of zeros. It will be assumed that the process always starts in a non-absorbing state. A PH distribution $\varphi$ is completely determined[3] by a pair $(\boldsymbol{u}, F)$ which is called the *representation* of the distribution. The probability distribution of $\varphi(.)$ for all $k \geq 1$ is given by

$$\begin{aligned} \varphi(k) &= \sum_{q_0 \in Q_T} P[\inf\{t \in \mathbb{N}^0 \mid X_t \in Q_A\} = k \mid X_0 = q_0].P[X_0 = q_0] \\ &= \boldsymbol{u}^T F^{k-1} \boldsymbol{e} \end{aligned} \tag{1.40}$$

which means that the probability of being absorbed in $k$ steps is the probability of starting in any transient state, then to move over transient states during $k-1$ steps, and finally to get absorbed. Each transient state of the representing MC is called a *phase*. This technique is powerful since it allows one to decompose complex distributions as a combination of phases. For instance, the class of PH distributions contains the negative binomial, the hyper-geometric and the discrete Coxian distributions, to name a few. These distributions can be instantiated using particular absorbing MC structures. This point is illustrated in figure 1.4. A distribution with an initial vector and a transition matrix with no particular structure is called here a *general PH distribution*.

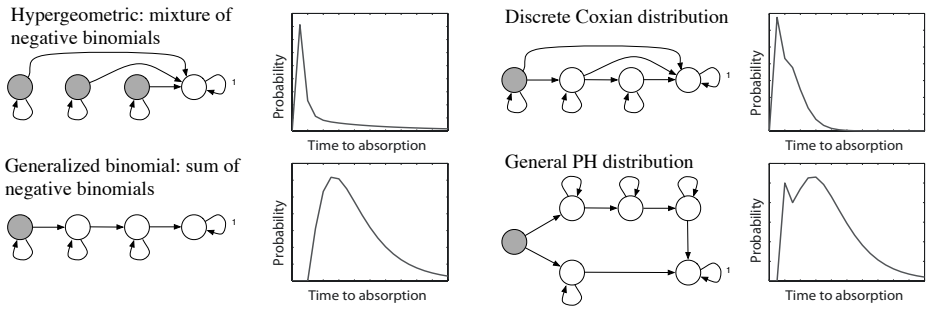



Figure 1.4: Different kinds of PH distributions and associate absorbing MC structures. The process has a strictly positive probability to start in states filled with gray and a null probability to start in the other states.

To illustrate phase-type distributions, let us consider the leaky bucket algorithm. The leaky bucket algorithm aims at detecting if a transmission channel between two hosts should be considered as unreliable. The error rate on the channel is assumed to be fixed and equal to $p$ with $0 < p < 1$. The receiver has a counter $c$ which is incremented by 1 when he receives

[3]Since the matrix $A$ is stochastic, the vector $\boldsymbol{e}$ can be obtained from the matrix $T$.

a corrupted packet and decremented by 1, provided $c > 0$, if the packet is valid. When the counter $c$ is equal to a threshold value $U$, the channel is discarded. The distribution of the number of packets transmitted on the channel before getting discarded can be modeled by a phase-type distribution. Let us define a reduced absorbing MC $T_{leaky} = \langle Q, A, \boldsymbol{\iota} \rangle$ where the state set is $Q = \{0, 1 \ldots, U\}$, *i.e.* each state corresponds to a counter value. The initial vector is $\boldsymbol{\iota}^T = (1, 0, \ldots, 0)$ and the transition matrix is (not explicitly specified elements are null):

$$A = \begin{pmatrix} 1-p & p & & & & \\ 1-p & 0 & p & & & \\ & \ddots & \ddots & \ddots & & \\ & & 1-p & 0 & p & \\ & & & & 1 & \end{pmatrix} \tag{1.41}$$

This absorbing MC is depicted in figure 1.5. Initially the process starts in state 0 as no corrupted packet has been received yet. Receiving a corrupted packet (with probability $p$) moves the process to the next state to the right while the reception of a valid packet (with probability $1-p$) moves the process to the previous state to the left (except for state 0). When the process reaches state $U$, it is absorbed. Hence, the number of packets transmitted on the channel before getting discarded is drawn from a phase-type distribution represented by the transient initial vector and the phase generator of the reduced absorbing MC $T_{leaky}$.

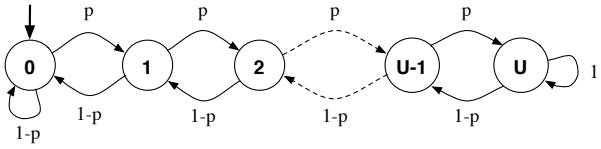



Figure 1.5: The leaky bucket algorithm modeled by a reduced absorbing MC. The channel bit error rate is constant and equal to $p$. Each state of the MC corresponds to a counter value. When the counter reaches the threshold value $U$ (*i.e.* the process is absorbed), the channel is discarded.

## 1.5 Estimation techniques

The present section deals with the estimation of MC from a data sample made of sequences defined over a discrete alphabet $\Sigma$. In this context, the set $Q$ of states of chain to estimate will directly correspond to the alphabet

$\Sigma$. In other words, one wants to estimate the parameters of an unknown MC from a set of random walks (the sequences) performed on this MC. In practice, the observed sequences may have been drawn from a process which is not a MC (*i.e.* the Markov property is not satisfied). However, one can try to approximate such a process with a MC. This section presents the maximum likelihood estimate of a MC as well as smoothing techniques derived from a so-called back-off scheme. MC estimated according to the Kneser-Ney (KN) back-off scheme, presented hereafter, are used in order to report comparative results in the context of sequence classification (see chapter 7). For a more detailed presentation of these smoothing techniques as well as their theoretical foundations, the reader is referred to [75]. In the sequel, given a string $s \in \Sigma^*$, $|s|$ will denote the length of $s$ and $s_i$ will stand for the symbol in the $i$-th position, the first position being 0

In the language modeling literature a $N$-gram denotes a $N-1$ order MC. Estimate a $N$-gram model amounts to estimate the homogeneous conditional probabilities $P[\mathtt{a} \mid h] = P[X_t = \mathtt{a} \mid X_{t-1} = h_{N-2}, \ldots, X_{t-(N-1)} = h_0]$ for all $\mathtt{a} \in \Sigma$ and $h \in \Sigma^{N-1}$, where. A maximum likelihood estimate of these probabilities can be simply obtained from counts of substrings. In the sequel, the count $C(h)$ denotes the number of times the substring $h \in \Sigma^*$ occurs in the data sample. The notation $C(h, b)$ stands for the number of times the substring $h \in \Sigma^*$ is immediately followed by the symbol $b$ in the sample. The maximum likelihood estimator of $P[\mathtt{a} \mid h]$ is given by

$$\hat{P}[\mathtt{a} \mid h] = \begin{cases} \frac{C(h,\mathtt{a})}{C(h)} & \text{if } C(h) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{1.42}$$

In general, the number of parameters of a $N$-gram model is given by $|\Sigma|^N$. This exponential grow with the model order makes the maximum likelihood estimator unreliable for high order model for the two following reasons: (i) a large number of events (i.e substrings) may not appear in the sample (although, they are likely to happen in other samples), leading to zero probabilities in the estimated model (ii) even for observed events, an accurate estimate of the conditional probabilities would require a huge amount of observations, very unlikely to be available in practice. To circumvent this issue, one classically applies smoothing techniques. In general, smoothing a distribution amounts to better spread the probability mass onto the set of possible events.

For $N$-gram models, a well known smoothing technique is the so-called *back-off* scheme [72]. An improved back-off modeling, called here the KN back-off scheme [75], is summarized hereafter.

$$P[\mathtt{a}|h] = \begin{cases} \dfrac{C(h,\mathtt{a}) - d_c}{C(h)} + \Gamma(h)\hat{P}_{back}[\mathtt{a} \mid h] & \text{if } C(h,\mathtt{a}) > 0, \quad \text{(C1)} \\ \Gamma(h)\hat{P}_{back}[\mathtt{a} \mid h] & \text{if } C(h,\mathtt{a}) > 0 \text{ and } C(h) > 0, \quad \text{(C2)} \\ \hat{P}_{back}[\mathtt{a} \mid h] & \text{if } C(h) = 0, \quad \text{(C3)} \end{cases}$$

where

$$\Gamma(h) = \frac{|\{\mathtt{a} \mid C(h,\mathtt{a}) > 0\}| d_c}{C(h)} \tag{1.43}$$

and where $\hat{P}_{back}[.]$ is a back-off distribution and $d_c \in \mathbb{R}^+$ is a discounting factor.

The general idea of the scheme is to subtract a value from the counts of seen events $(h, \mathtt{a})$ such that $C(h, \mathtt{a}) > 0$. The discounted probability mass is spread to unseen events in proportion to their back-off estimates $\hat{P}_{back}[\mathtt{a} \mid h]$ (cases (C1) and (C2)). The factor $\Gamma(h)$ can be considered as a normalization factor in order to define proper conditional distributions, *i.e.* $\sum_{b \in \Sigma} P[\mathtt{a} \mid h] = 1, \forall h \in \Sigma^*$. Note also that $\Gamma(h)$ is only well-defined if $h$ occurs at least one time in the sample. Otherwise, the case (C3) applies and the back-off distribution is used in all cases.

Smoothing techniques derived from this scheme differ by the definitions of the discounting factor $d_c$ and the back-off probability $\hat{P}_{back}[\mathtt{a} \mid h]$. In the case of the absolute discounting [96], the discounted value $d_c$ is constant. In contrast, the Turing-Good discounting [72] and the Witten-Bell discounting [135] respectively define the discounted value $d_c$ according to the count $C(h, \mathtt{a})$ and to the number of distinct events.

The simplest case for the back-off distribution is to define $\hat{P}_{back}[\mathtt{a} \mid h] = \hat{P}[\mathtt{a} \mid h_{-1}]$ where $h_{-1}$ denotes the string $h$ trimmed by one symbol on the left. In other words, for a $N$-gram, the back-off distribution is simply a $(N-1)$-gram. It is important to note that this scheme is applied recursively down to the unigram (*i.e.* a 1-gram). Interestingly, a $N$-gram defines in this way actually forms a variable order MC. The order of this MC ranges from 0 to $N-1$. The base case of the recursion is the order 0 MC (*i.e.* the unigram): $P(\mathtt{a}) = C(\mathtt{a}) / \sum_{\mathtt{a} \in \Sigma} C(\mathtt{a})$, which is always strictly positive when $\mathtt{a}$ occurs at least one time in the sample. If this assumption is not satisfied, the base of the recursion becomes the uniform distribution $P(\mathtt{a}) = 1/|\Sigma|$. The KN back-off scheme, presented here, also uses the back-off distribution for observed events (second term of the case (C1)). The rationale is that the back-off distribution is more accurately estimated as it is less specific and thus relies on more observations. The resulting model is a mixture of Markov chains of various orders.

Another definition of the back-off distribution proposed by Kneser and Ney [75] is the following:

$$\hat{P}_{back}[\mathtt{a} \mid h] = \frac{C(., h_{-1}, \mathtt{a})}{\sum_{\mathtt{a}' \in \Sigma} C(., h_{-1}, \mathtt{a})} \tag{1.44}$$

where

$$C(., h_{-1}, \mathtt{a}) = |\{\mathtt{b} \mid C(\mathtt{b}h_{-1}, \mathtt{a}) > 0\}| \tag{1.45}$$

Here, $C(., h_{-1}, \mathtt{a})$ denotes the number of distinct events $(., h_{-1})$ preceding $\mathtt{a}$ in the sample. It should be noted that the frequencies of these events are not taken into account. It was shown experimentally that this back-off distribution offers a better performance than $\hat{P}_{back}[\mathtt{a} \mid h] = \hat{P}[\mathtt{a} \mid h_{-1}]$ in the context of sequence prediction, *i.e* the prediction of the next symbol of a sequential process knowing its past [75].

# Chapter 2

# Hidden Markov Models

*This chapter reviews the theory and classical algorithms related to Hidden Markov Models (HMMs). The second part of this thesis is concerned with the structural estimation of such models. HMMs extend MC by adding a probabilistic emission function onto the states of the model. Consequently, the process outcomes no longer correspond to the states but to output symbols belonging to a discrete or continuous alphabet. These models are called* hidden*, since in general, it is not possible to know with certainty the states reached by the process by looking at its outcomes. From a theoretical viewpoint HMMs are equivalent to Probabilistic Non-deterministic Finite state Automata (PNFA). Consequently, learnability results as well induction techniques (see chapter 4) related to one of these two formalisms also apply to the other. Practically, HMMs are widely used in many pattern recognition areas, including biological sequence modeling [46], speech recognition [108], optical character recognition [83] and information extraction [52], to name a few. There are three main problems concerning HMMs: (i) compute the likelihood of a sequence with respect to a given model (i.e. the probability that the model has generated the sequence), (ii) given a sequence, compute the best path (i.e. sequence of states) in the model to generate/observe this sequence, and (iii) learn a HMM from a data sample. While there are efficient algorithms to solve the two first problems, the learning problem remains hard to tackle in practice. Learning a HMM from data is much harder than learning a MC since the process is only partially observed (the state identities are not provided in the sample). Moreover, the learning process does not only amount to estimate the model parameters but also its structure, i.e. its transition graph. The learning problem is restricted here to parameter estimation and the most traditional approach, known as the Baum-Welch or forward-backward algorithm, is briefly reviewed. Chapter 5 proposes an original algorithm to learn the structure of a HMM from a data sample. This chapter is organized as follows. Section 2.1 reviews ba-*

*sic definitions about HMMs. Next, section 2.2 highlights the links between HMMs and probabilistic automata. Finally, section 2.3 presents the three main problems concerning HMMs and their classical solutions. For a more detailed introduction to HMMs, the reader is referred to the classical Rabiner tutorial [107]. The paper [45] thoroughly studies the links between HMMs and probabilistic automata.*

## 2.1 Definitions

A HMM defines a stochastic process $\{O_t \in \Sigma\}_t$ where $\Sigma$ is a discrete or continuous alphabet. In this work, we concentrate our attention on HMMs defined over a discrete alphabet of symbols. In contrast with MC, a HMM needs not to focus on local dependencies in a process. In other words, the probability of an outcome can potentially depend on an unbounded history of past events. HMMs are however closely related to MC and were initially studied as "functions of finite Markov chains" [12, 8]. The classical definition of an order 1 discrete HMM is provided hereafter. Many variants can be found in the literature, including higher order HMMs, models with emission on transitions (see definition 27) or with a continuous emission density, typically defined by a Gaussian or a multi-Gaussian instead of a discrete (multinomial) distribution (see, for example, [103], [107], [108], [66]).

**Definition 20** *An order 1 discrete* Hidden Markov Model (HMM) *is a 5-tuple* $H = \langle \Sigma, Q, A, B, \iota \rangle$ *where* $\Sigma$ *is an alphabet,* $Q$ *is a set of states,* $A : Q \times Q \to [0,1]$ *is a mapping defining the probability of each transition,* $B : Q \times \Sigma \to [0,1]$ *is a mapping defining the emission probability of each letter on each state, and* $\iota : Q \to [0,1]$ *is a mapping defining the initial probability of each state. The following properness constraints must be satisfied:*

- $\forall q \in Q, \sum_{q' \in Q} A(q,q') = 1$
- $\forall q \in Q, \sum_{\mathtt{a} \in \Sigma} B(q,\mathtt{a}) = 1$
- $\sum_{q \in Q} \iota(q) = 1$

As only finite HMMs are considered here, the transition function $A(.,.)$ can be represented by a $|Q| \times |Q|$ matrix $A$ such that $A_{qq'} = A(q,q')$, the emission function $B(.,.)$ can be represented by a $|Q| \times |\Sigma|$ matrix $B$ such that $B_{q\mathtt{a}} = B(q,\mathtt{a})$ and the initialization function $\iota(.)$ can be represented by a $|Q|$-dimensional vector $\boldsymbol{\iota}$ such that $\iota_q = \iota(q)$. Properness constraints impose that the vector $\boldsymbol{\iota}$ and each line of the matrices $A$ and $B$ must be stochastic.

To illustrate this, let us consider the following example. During holidays, in the Europe capital (where the weather evolves according to the MC

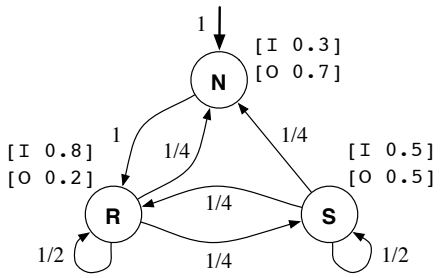



Figure 2.1: A HMM example modeling the sequence of places (indoor (I) or outdoor (O)) visited by a PhD student during holidays in the Europe capital.

$T_{\text{Weather}}$ presented in chapter 1), a PhD student has 7 out of 10 chances to spend the day outdoor if the weather is nice (N), 8 out of 10 chances to spend the day indoor if it is raining (R) and he has an even chance to spend the day indoor/outdoor if it is snowing (S). The sequence of places (indoor/outdoor) visited by the PhD student can be modeled by a HMM $H_{\mathtt{I/O}}$ with an alphabet $\Sigma = \{\mathtt{I}, \mathtt{O}\}$ (I: indoor; O: outdoor), a state set $Q = \{\text{N,R,S}\}$, an initial vector $\boldsymbol{\iota} = (1, 0, 0)$ and the following transition and emission matrices:

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 1/4 & 1/2 & 1/4 \\ 1/4 & 1/4 & 1/2 \end{pmatrix}; \quad B = \begin{pmatrix} 0.3 & 0.7 \\ 0.8 & 0.2 \\ 0.5 & 0.5 \end{pmatrix}$$

This HMM is displayed in Figure 2.1. In this example, the hidden aspect is that one assumes that we can only observe if the PhD student was indoor or outdoor during a particular day but not the weather on that day. In this sense, the *underlying MC* is hidden by the *emission process*.

**Definition 21 (Underlying MC of a HMM)** *Given a HMM $H = \langle \Sigma, Q, A, B, \iota \rangle$, the* underlying Markov chain *$T$ is the 3-tuple $\langle Q, A, \boldsymbol{\iota} \rangle$.*

The process $\{X_t \in Q\}_t$ associated to the underlying MC will be called the *hidden process*. That is, the random variable $X_t$ provides the state reached at time $t$. This process evolves as in definition 3. The *emission process* is formally defined as follows.

**Definition 22 (Emission process)** *Given a HMM $H = \langle \Sigma, Q, A, B, \iota \rangle$, let $q_t$ denotes the state reached by the hidden process at time $t \in \mathbb{N}$, i.e. $X_t = q_t$. The* emission process *$\{O_t \in \Sigma\}_t$ consists of emitting a symbol $\mathtt{a} \in \Sigma$ according to the emission function $B(q_t, .)$ at each time $t \in \mathbb{N}$. Running such a process up to a time $t \in \mathbb{N}$ results in a sequence of $t + 1$ symbols.*

An instance of the emission process will be called a *HMM random walk*. It is important to note that all the notions presented in chapter 1 concerning MC also apply to the underlying MC of HMMs. These notions are very useful to analyze the behavior of HMMs. For instance, the stationary distribution of the underlying regular MC $T_{\text{Weather}}$ is $\boldsymbol{\pi} = (N : 0.2, R : 0.53, S : 0.27)$. Consequently, the expected proportion of indoor days is given by $(0.2 * 0.3) + (0.53 * 0.8) + (0.27 * 0.5) = 0.62$.

During a random walk in a HMM, the sequence of states reached by the hidden process is called a *HMM path*.

**Definition 23 (HMM path)** *Let $H = \langle \Sigma, Q, A, B, \iota \rangle$ be a HMM. A* path *in $M$ is a word defined on $Q^*$. For any sequence $s \in \Sigma^*$ and any path $\nu \in Q^*$, the probabilities $P_H(s, \nu)$ and $P_H(s)$ are defined as follows:*

$$P_H(s,\nu) = \begin{cases} \iota(\nu_0)B(\nu_0, s_0)\prod_{i=1}^{|s|-1} A(\nu_{i-1}, \nu_i)B(\nu_i, s_i) \text{ if } |s| = |\nu| > 0, \\ 1 \text{ if } |s| = |\nu| = 0 \text{ and} \\ 0 \text{ otherwise.} \end{cases}$$

$$P_H(s) = \sum_{\nu \in Q^*} P(s, \nu).$$

$P_H(s, \nu)$ is the probability to emit the sequence $s$ while following path $\nu$. Along any path, the emission process is Markovian since the probability of emitting a letter on a given state only depends on that state. HMMs are used to model processes for which the existence of such a path (or state sequence) can be assumed while the actual states are not observed. $P_H(s)$ can be interpreted as the probability of generating/observing a finite sequence $s$ as part of a random walk through the model. For instance, the sequence $s = \texttt{IOI}$ can have been generated by the HMM $H_{\texttt{I/O}}$ along the following paths: NRI, NRR or NRS. Consequently, the probability (or the likelihood) of the sequence $s = \texttt{IOI}$ with respect to $H_{\texttt{I/O}}$ is $P_{H_{\texttt{I/O}}}(\texttt{IOI}) = (1*0.3)*(1*0.2)*(0.25*0.3) + (1*0.3)*(1*0.2)*(0.5*0.8) + (1*0.3)*(1*0.2)*(0.25*0.5) = 0.036$. Section 2.3.1 presents an efficient algorithm to compute such a likelihood.

The structure or the topology of an HMM can be characterized by the transient/final class partition of its underlying MC (see chapter 1). Hence, we will use the same denominations for HMMs as for MC. Irreducible HMMs are often called ergodic HMMs in the literature. Sometimes, an ergodic HMM also denotes a fully connected HMM. Up to our knowledge, the distinction between a cyclic HMM and a regular HMM is not well-known in the HMM literature. A popular topology used in automated speech recognition is the *left-to-right* structure. The left-to-right structure denotes a reduced

absorbing HMM (the last state being absorbing) such that a state with index $q$ can only be connected to states with an index $q' \geq q$. Furthermore, the process starts in the first state (the state with the lowest index) with probability 1. When the states are displayed linearly according to the increasing order of their indices, the process starts in the leftmost state, then it moves to the right (or to itself) until it ends up in the absorbing state. This kind of topology can adequately model sequences whose properties are changing over time. A 5-states left-to-right structure is depicted in Figure 2.2. In this example, the process cannot move to the right by more than two states at once.

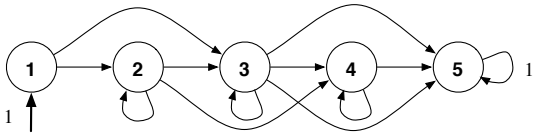



Figure 2.2: A left-to-right HMM structure.

The transition matrix of such a left-to-right HMMs is upper triangular. According to the application domain, many relevant topologies can be designed. One of the objectives of this thesis is to provide a procedure to efficiently learn the structure of a HMM from data (see chapter 5).

## 2.2 Links between PA and HMMs

This section highlights the connections between HMMs and probabilistic languages (also known as stochastic languages) represented by probabilistic automata (PA). We review here some of the results thoroughly studied in [45]. A probabilistic language is a distribution defined over sequences.

**Definition 24** *A* distribution *or* probabilistic language $\Psi$ *over* $\Sigma^*$ *is a function* $\Psi : \Sigma^* \rightarrow [0, 1]$ *satisfying* $\sum_{s \in \Sigma^*} \Psi(s) = 1$

Importantly, HMMs are equivalent, in the sense that they represent the same class of probabilistic languages, to a particular class of PA called Probabilistic Non-deterministic Finite state Automata with no final probabilities (PNFA).

**Definition 25** *A* probabilistic Non-deterministic finite automaton with no final probabilities *(PNFA) is a 4-tuple* $\langle \Sigma, Q, \phi, \iota \rangle$ *where* $\Sigma$ *is a finite alphabet,* $Q$ *is a finite set of states,* $\phi : Q \times \Sigma \times Q \rightarrow [0, 1]$ *is a mapping defining the transition probability function and* $\iota : Q \rightarrow [0, 1]$ *is a mapping defining*

*the initial probability of each state. The following properness constraints must be satisfied:*

$$\sum_{q\in Q} \iota(q) = 1 \text{ and } \forall q \in Q, \sum_{\mathtt{a}\in\Sigma} \sum_{q'\in Q} \phi(q, \mathtt{a}, q') = 1.$$

The probability of generating a sequence[1] $s \in \Sigma^*$ in such a model is defined as follows.

**Definition 26** *Let $NA = \langle \Sigma, Q, \phi, \iota \rangle$ be a PNFA. The functions $P_{NA} : \Sigma^* \to [0, 1]$ is defined as follows:*

$$P_{NA}(s) = \sum_{q,q'\in Q} \iota(q)\phi(q, s, q')$$

*where $\phi$ is an extension of the transition function defined on $Q \times \Sigma^* \times Q$ such that*

- $\phi(q, \varepsilon, q') = \begin{cases} 1 \text{ if } q = q' \\ 0 \text{ otherwise} \end{cases}$
- $\forall s \in \Sigma^*, \forall \mathtt{a} \in \Sigma, \phi(q, s\mathtt{a}, q') = \sum_{q''\in Q} \phi(q, s, q'')\phi(q'', \mathtt{a}, q')$

*where $\epsilon$ denotes the empty sequence.*

A PNFA (equivalently, a HMM) defines a distribution (*i.e.* a language) for each sequence length, *i.e.* $\sum_{s\in\Sigma^l} P_{NA}(s) = 1$ for all $l \in \mathbb{N}$. More precisely, a probabilistic language is obtained for any complete prefix-free set of $\Sigma^*$, that is, a subset $U \subseteq \Sigma^*$ such that (i) a sequence in $U$ cannot be a prefix of another sequence in $U$ and (ii) each sequence in $\Sigma^*$ has a prefix in $U$ or is a prefix of a sequence in $U$ (for details, see [45]).

Let us show how to convert a HMM into an equivalent PNFA and conversely. First, let us introduce an intermediate kind of models called hidden Markov models with emissions on transitions (HMMT) used in the conversions HMM $\leftrightarrow$ PNFA.

**Definition 27** *A discrete* Hidden Markov Model with transition emission (HMMT) *is a 5-tuple $\langle \Sigma, Q, A, B, \iota \rangle$, where $\Sigma$ is an alphabet, $Q$ is a set of states, $A : Q \times Q \to [0, 1]$ is a mapping defining the probability of each transition, $B : Q \times \Sigma \times Q \to [0, 1]$ is a mapping defining the emission probability of each letter on each transition, and $\iota : Q \to [0, 1]$ is a mapping defining the initial probability of each state. The following properness constraints must be satisfied:*

[1] More precisely, since we consider PNFA without final probabilities, the probability $P_{NA}(s)$ given in definition 26 denotes the probability of generating a (possibly infinite) sequence with finite prefix $s$.

- $\forall q \in Q, \sum_{q' \in Q} A(q, q') = 1$
- $\forall q, q' \in Q, \sum_{\mathtt{a} \in \Sigma} B(q, \mathtt{a}, q') = \begin{cases} 1 \text{ if } A(q, q') > 0 \\ 0 \text{ otherwise.} \end{cases}$
- $\sum_{q \in Q} \iota(q) = 1.$

For such models, the probability of generating a (prefix) sequence $s \in \Sigma^*$ is defined hereafter.

**Definition 28** *Let $H = \langle \Sigma, Q, A, B, \iota \rangle$ be a HMMT. A* path *in $H$ is a word defined on $Q^*$. For any sequence $s \in \Sigma^*$ and any path $\nu \in Q^*$, the probabilities $P_H(s, \nu)$ and $P_H(u)$ are defined as follows:*

$$P_H(s, \nu) = \begin{cases} \iota(\nu_0) \prod_{i=1}^{|s|} [B(\nu_{i-1}, s_{i-1}, \nu_i) A(\nu_{i-1}, \nu_i)] \text{ if } |\nu| = |s| + 1, \text{ and} \\ 0 \text{ otherwise,} \end{cases}$$

$$P_H(s) = \sum_{\nu \in Q^*} P(s, \nu).$$

Next, we show how to transform a PNFA into an equivalent HMMT. The resulting HMMT can be transformed into an equivalent HMM.

**Proposition 1** *Let $NA = \langle \Sigma, Q, \phi, \iota \rangle$ be a PNFA with no final probabilities. There exists an equivalent HMMT $H = \langle \Sigma, Q, A, B, \iota \rangle$.* **Proof** *see [45].*

The transformation can be done as follows. $\Sigma$, $Q$ and $\iota(.)$ are identical for $NA$ and $H$. The transition functions for $H$ are defined as follows.

- $\forall q, q' \in Q, A(q, q') = \sum_{\mathtt{a} \in \Sigma} \phi(q, \mathtt{a}, q')$
- $\forall q, q' \in Q, \forall \mathtt{a} \in \Sigma, B(q, \mathtt{a}, q') = \begin{cases} \frac{\phi(q, \mathtt{a}, q')}{\sum_{\mathtt{a} \in \Sigma} \phi(q, \mathtt{a}, q')} & \text{if } \sum_{\mathtt{a} \in \Sigma} \phi(q, \mathtt{a}, q') > 0 \\ 0 & \text{otherwise.} \end{cases}$

This transformation is illustrated in Figure 2.3.

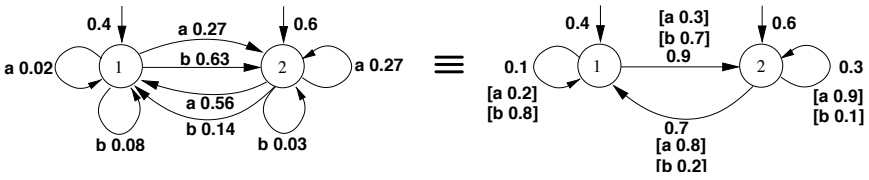



Figure 2.3: The conversion of a PNFA (left) into an equivalent HMMT (right).

**Proposition 2** *Let $H = \langle \Sigma, Q, A, B, \iota \rangle$ be a HMMT, there exists an equivalent HMM $H' = \langle \Sigma, Q', A', B', \iota' \rangle$.* **Proof** *see [28].*

$H'$ is defined as follows.

- $Q' = \{(q, q') \in Q \times Q | A(q, q') > 0\}$. The states of $Q'$ represents pairs of states in $Q$ that are connected by a strictly positive transition probability.
- $\forall (q, q'), (q'', q''') \in Q', A((q, q'), (q'', q''')) = \begin{cases} A(q'', q''') & \text{if } q' = q'', \\ 0 & \text{otherwise.} \end{cases}$
- $\forall (q, q') \in Q', \forall \mathtt{a} \in \Sigma, B((q, q'), \mathtt{a}) = B(q, \mathtt{a}, q')$
- $\forall (q, q') \in Q', \iota'((q, q')) = \iota(q) A(q, q')$

Let us note that the number of states $|Q'|$ is less or equal to $|Q|^2$. This transformation is illustrated in Figure 2.4.

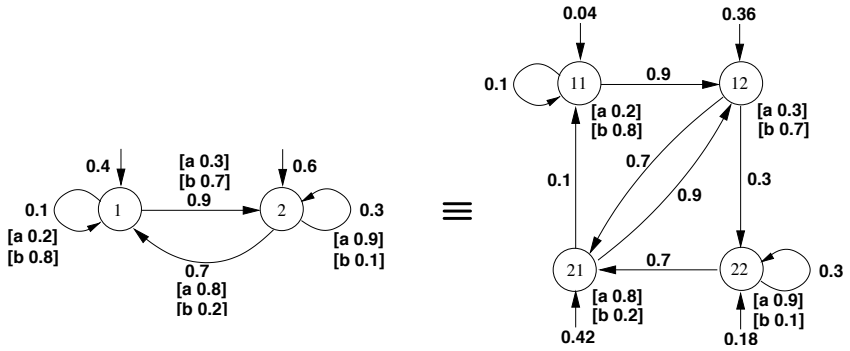



Figure 2.4: The conversion of a HMMT (left) into an equivalent HMM (right).

The two preceding results show that a HMM generating the same distributions as a given PNFA can be constructed in two steps. Conversely, a PNFA equivalent to a given HMM can be obtained directly.

**Proposition 3** *Let $H = \langle \Sigma, Q, A, B, \iota \rangle$ be a HMM. There exists an equivalent PNFA with no final probabilities $NA = \langle \Sigma, Q, \phi, \iota \rangle$.* **Proof** *see [45].*

$\Sigma$, $Q$ and $\iota(.)$ are the same for $H$ and $NA$. The transition function of $NA$ is defined as follows:

$$\forall q, q' \in Q, \forall \mathtt{a} \in \Sigma, \phi(q, \mathtt{a}, q') = B(q, \mathtt{a}) A(q, q')$$

This transformation is illustrated in Figure 2.5.

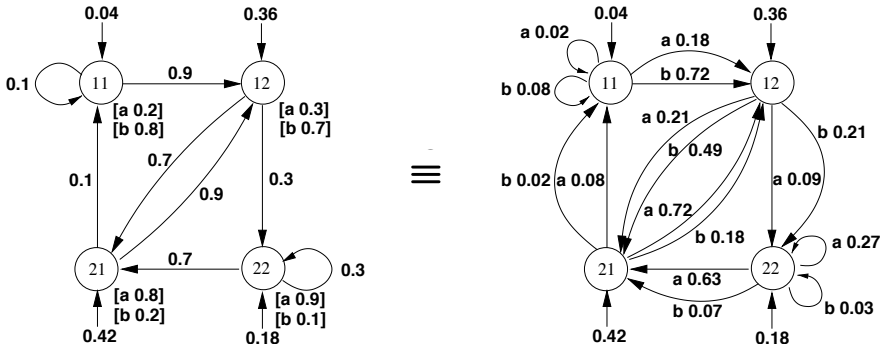



Figure 2.5: The conversion of a HMM (left) into an equivalent PNFA (right).

Note that the PNFA depicted in the right part of Figure 2.5 is equivalent to the PNFA displayed in the left part of figure 2.3, since it results from the successive transformations described in propositions 1, 2 and 3. These two PNFA generate the same distributions. They are however structurally distinct. The question of determining if two structurally distinct models are equivalent is studied in [7]. In a few words, [7] proposes an algorithm to test if two HMMs $H = \langle \Sigma, Q, A, B, \iota \rangle$ and $H' = \langle \Sigma, Q', A', B', \iota' \rangle$ are equivalent, in computational time $\mathcal{O}(|\Sigma|(|Q| + |Q'|)^3)$. The same work also proposes a technique to reduce a HMM $H = \langle \Sigma, Q, A, B, \iota \rangle$ into a canonical Generalized Markov Model (GMM), *i.e.* a HMM without the positivity constraint on the parameters, in computational time $\mathcal{O}(|\Sigma||Q|^3)$. An important consequence of the equivalence between HMMs and PNFA is that learnability results as well as reduction and induction techniques (see chapter 4) related to one of these two formalisms also apply to the other. A similar equivalence can be obtained when considering PNFA and HMMs with final probabilities. In this case, both models represent the class of probabilistic regular languages [45].

## 2.3 The three main problems and their classical solutions

Modeling a process often aims at performing predictions of its future outcomes. One can also analyze the model in order to discover relevant properties of the process under study. To do so, there are three basic problems that need to be solved:

1. Given a sequence $s \in \Sigma^*$ and a HMM $H = \langle \Sigma, Q, A, B, \iota \rangle$, how to compute efficiently the sequence likelihood $P_H(s)$?

2. Given a sequence $s \in \Sigma^*$ and a HMM $H = \langle \Sigma, Q, A, B, \iota \rangle$, how to obtain the HMM path that best explains the emission of $s$ in $H$?

3. Given a sequence $s \in \Sigma^*$, how to estimate the probabilistic parameters of a HMM $H = \langle \Sigma, Q, A, B, \iota \rangle$ such that the sample likelihood $P_H(s)$ is maximized?

The classical solutions to these problems are reviewed hereafter.

### 2.3.1 Computing the sequence likelihood

Computing the likelihood of a sequence $s$ with respect to a HMM $H = \langle \Sigma, Q, A, B, \iota \rangle$ is particularly useful in sequence classification problems. In this setting, we have a set of $n$ examples $\{(s^1, y_1), \ldots, (s^n, y_n)\}$ where $s^i \in \Sigma^*$ is a sequence and $y_i \in \mathcal{Y}$ its label (or its class), and we want to predict the label of new sequences. For each class $y$, a model is estimated (see section 2.3.3) using the sequences labeled $y$. To classify a new sequence $s$, one computes the likelihood of $s$ for each estimated model. According to the maximum likelihood decision rule, the predicted label is the label associated to the model offering the largest likelihood. Alternatively, one can take the class priors into account in a maximum *a posteriori* decision scheme (see chapter 6). A naive way to compute the sequence likelihood $P_H(s)$ is to sum the probability of generating $s$ over all possible paths as in definition 23. However, the number of paths is, in general, exponential in the size of the sequence $|s|$, leading to an intractable algorithm. The sequence likelihood can be efficiently computed using the forward or the backward recurrences. These recurrences are also the basis of the HMM estimation procedure (see section 2.3.3). Considering a sequence $s = s_0 \ldots s_l$ (*i.e.* $l = |s| - 1$), the forward variables are defined as follows :

$$\begin{aligned} \alpha(q,t) &= P[O_t = s_t, \ldots, O_0 = s_0, X_t = q \mid H] && (2.1) \\ &= P[s_{:t}, X_t = q \mid H] && (2.2) \end{aligned}$$

for all $q \in Q$ and for all $0 \leq t \leq l$. That is, $\alpha(q,t)$ computes the probability that the model $H$ generates the sequence $s$ up to position $t$ (denoted by $s_{:t}$), while reaching state $q$. One can determine these variables by computing the following recurrence:

$$\begin{aligned} \text{(Base case)} \quad \alpha(q,0) &= \iota(q)B(q,s_0) && (2.3) \\ \text{(Induction)} \quad \alpha(q,t) &= [\sum_{q' \in Q} \alpha(q', t-1)A(q',q)]B(q,s_t) && (2.4) \end{aligned}$$

for all $q, q' \in Q$ and for all $0 < t \leq l$. The base case, $\alpha(q,0)$, is the probability to start in $q$ and to emit the symbol $s_0$. The probability to

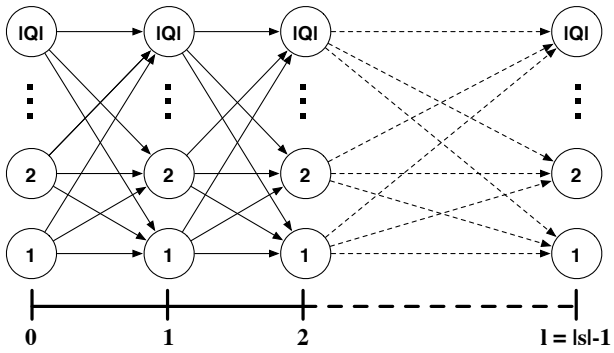



Figure 2.6: The lattice used to compute efficiently the forward and backward recurrences. The bottom line displays the discrete time indices.

generate $s_{:t}$ while reaching state $q$ is the sum over all states $q' \in Q$ of the probabilities to generate $s_{:t-1}$ while reaching state $q'$ then to move from $q'$ to $q$ and finally to emit symbol $s_t$. By the total probability theorem, the sequence likelihood can be reformulated as follows

$$P_H(s) = \sum_{q \in Q} P[s \mid X_l = q, H]\ P[X_l = q \mid H] \tag{2.5}$$

$$= \sum_{q \in Q} P[s, X_l = q \mid H] = \sum_{q \in Q} \alpha(q, l) \tag{2.6}$$

The forward recurrence can be efficiently computed using the lattice displayed in Figure 2.6. The computation is made from left to right since the variables $\alpha(., t)$ only depends on the variables $\alpha(., t-1)$. The time complexity of this computation is $\Theta(|s|\ n_t)$ where $n_t$ is the number of transitions in the model. An equivalent upper bound of the computational time is $\mathcal{O}(|s||Q|^2)$ which is tight for dense models.

Either the forward or backward recurrence can be used to compute a sequence likelihood. However, both recurrences are required to estimate efficiently the model parameters as detailed in section 2.3.3. The backward variables $\beta(q, t)$ are defined hereafter.

$$\beta(q, t) = P[O_l = s_l, \ldots, O_{t+1} = s_{t+1}, X_t = q \mid H] \tag{2.7}$$

$$= P[s_{t+1:}, X_t = q \mid H] \tag{2.8}$$

for all $q \in Q$ and for all $0 \leq t \leq l$. That is $\beta(q, t)$, is the probability to generate $s$ from position $t+1$ to the end (denoted by $s_{t+1:}$), while leaving from state $q$. These variables can be computed using the following recurrence:

$$\begin{aligned} \text{(Base case)} \quad \beta(q,l) &= 1 && (2.9)\\ \text{(Induction)} \quad \beta(q,t) &= \sum_{q' \in Q} A(q,q')B(q',s_{t+1})\beta(q',t+1) && (2.10) \end{aligned}$$

for all $q \in Q$ and for all $0 \leq t < l$. Initially, the backward variables $\beta(.,l)$ are set to one[2]. The probability to generate $s_{t+1:}$ while leaving from $q$ is the sum over all states $q' \in Q$ of the probability to move from $q$ to $q'$, then to emit $s_{t+1}$ and finally to generate $s_{t+2:}$ from $q'$. Using the backward variables, the sequence likelihood is provided by

$$P_H(s) = \sum_{q \in Q} \iota(q)B(q,s_0)\beta(q,0) \tag{2.11}$$

The backward recurrence can also be computed in a time complexity $\Theta(|s|\ n_t)$ using a similar lattice as for the forward recurrence computation.

### 2.3.2 Computing the optimal path

The objective is to find the HMM path that best explains (according to some optimality criterion) the generation of a sequence $s \in \Sigma^*$ in the model. A first optimality criterion requires that the states of the path are individually the most likely. To be more precise, let us compute the probability to be in state $q$ at time $t$ $(0 \leq t \leq l)$ while generating the sequence $s = s_0 \dots s_l$:

$$P[X_t = q \mid s, H] = \frac{\alpha(q,t)\beta(q,t)}{\sum_{q' \in Q} \alpha(q',t)\beta(q',t)} \tag{2.12}$$

That is $\alpha(q,t)$ accounts for the generation of $s_{:t}$ while reaching state $q$ and $\beta(q,t)$ accounts for the emission of the remainder of the sequence while leaving from $q$. At time index $t$, the individually most likely state is defined as follows:

$$q_t = \arg\max_{q \in Q}(P[X_t = q \mid s, H]) \tag{2.13}$$

Unfortunately, this criterion possibly results in invalid paths. Indeed, such a path could contains two consecutive states $q$ and $q'$ which are not adjacent, *i.e.* $A_{qq'} = 0$. The rationale is that this criterion optimizes the probability of an individual state at each time index but not the probability of the sequence of states. Optimizing the probability of the sequence of states, *i.e.* $P[X_l = q_l, \dots, X_0 = q_0 \mid s, H]$, can be performed using a method based on

[2]Indeed, there is a probability 1 to generate an empty sequence from any state $q \in Q$ (see definition 23).

dynamic programming called the Viterbi algorithm [49]. First, let us define the maximal probability to generate $s_{:t}$ along a path ending by state $q$.

$$\delta(q,t) = \max_{q_0,\ldots,q_{t-1}} P[s_{:t}, X_t = q, X_{t-1} = q_{t-1}, \ldots, X_0 = q_0, \mid H] \tag{2.14}$$

By induction on $t$, one obtains:

$$\begin{aligned} \delta(q,t) &= \delta(\nu(q,t), t-1)\ A(\nu(q,t), q)\ B(q, s_t) & (2.15)\\ \nu(q,t) &= \arg\max_{q' \in Q} [\delta(q', t-1) a(q', q)] & (2.16) \end{aligned}$$

for all $0 < t \leq l$ and for all $q \in Q$. The $\nu(q,t)$ variable provides the state $q'$ reached at time $t-1$ while generating $s_{:t-1}$, that maximizes the probability to be in state $q$ at time $t$. Using these recurrences, the optimal path[3] $q_0^{opt}, \ldots, q_l^{opt}$ is computed by the following procedure:

1. **Initialization**, for all $q \in Q$:

$$\begin{aligned} \delta(q,0) &= \iota(q) B(q, s_0) & (2.17)\\ \nu(q,0) &= \emptyset & (2.18) \end{aligned}$$

2. **Recursion**, for all $0 < t \leq l$ and for all $q \in Q$:

$$\begin{aligned} \delta(q,t) &= \delta(\nu(q,t), t-1)\ A(\nu(q,t), q)\ B(q, s_t) & (2.19)\\ \nu(q,t) &= \arg\max_{q' \in Q} [\delta(q', t-1) a(q', q)] & (2.20) \end{aligned}$$

3. **Termination**:

$$\begin{aligned} p^{opt} &= \max_{q \in Q} \delta(q, l) & (2.21)\\ q_l^{opt} &= \arg\max_{q' \in Q} [\delta(q', l)] & (2.22) \end{aligned}$$

4. **Path recovery**, for all $0 \leq t < l$:

$$q_t^{opt} = \nu(q_{t+1}^{opt}, t+1) \tag{2.23}$$

where $p^{opt}$ is the probability of generating sequence $s$ along the best path. The Viterbi algorithm is similar to the forward recurrence computation except for the summation in equation (2.10) which is replaced by a maximization in equations (2.19)-(2.20). Consequently, the Viterbi algorithm can be efficiently computed using the same lattice as for the forward recurrence. Additionally, a backtracking step (*i.e.* path recovery) is required to recover the optimal state sequence. The time complexity of the Viterbi algorithm is $\Theta(|s|\ n_t)$ where $n_t$ is the number of transitions in the model.

[3]The optimal path is also known as the Viterbi path.

### 2.3.3 Estimating the model parameters

The objective is to estimate the parameters of a HMM from a training sequence $s = s_0 \dots s_l$ drawn from some unknown process of interest called here the *target process*. More precisely, one wants to adjust the model parameters such that the sequence likelihood is maximized. It is important to note that the structure of the model is also crucial in order to accurately model the target process. The current section only deals with the estimation of parameters assuming a predefined structure. The second part of this thesis focus on the structural estimation of HMMs.

Estimating the parameters of a HMM appears to be a difficult problem since the likelihood function is highly non-linear and non-convex. Instead of trying to solve these non-linear equations[4], Baum et al. have set up an iterative hill-climbing procedure, the Baum-Welch algorithm, which guarantees that the likelihood increases at each iteration and that it converges to some local minimum[5] of the likelihood function. Gradient-based techniques are an alternative to Baum-Welch in order to maximize the likelihood. There is a strong connection between the Baum-Welch algorithm and gradient-based techniques since the Baum-Welch algorithm can be viewed as a gradient-ascent technique with an automatic update rate selection [113]. Another advantage of the Baum-Welch algorithm is that properness constraints are automatically satisfied while additional constraints must be added when using gradient-based methods.

The Baum-Welch algorithm is an instance of the Expectation-Maximization (EM) algorithm [36]. In a few words, EM finds the maximum likelihood estimates of the parameters $\rho$ of a joint distribution $P[O, \mathcal{H}|\rho]$ when the variable $\mathcal{H}$ is unobserved ($\mathcal{H}$ is a *latent* or *hidden* variable). In our case, the observed variables are the symbols generated by the emission process $\{O_t\}_t$ and the hidden variables are the states reached by the hidden process $\{X_t\}_t$ also called the hidden path. The parameters to estimate are the initial function $\iota(.)$, the transition function $A(.,.)$ and the emission function $B(.,.)$. EM computes iteratively the parameters $\rho$ that maximize $\mathbb{E}\left[\log(P[O, \mathcal{H} \,|\, \rho]) \,|\, O\right]$. Each EM iteration involves two steps: (i) the computation of the conditional expectation of the latent variables given the last parameter values $\rho^{t-1}$ and the partial observations $O_t$, (ii) the maximization of the parameters $\rho^t$ given the conditional expectation of the latent variables.

[4] We are referring here to the KKT optimality conditions merely stating that the partial derivatives of the Lagrangian with respect to the model parameters must be zero at the optimum.

[5] In fact, a global optimum of the likelihood can be reached but there is no guaranty.

**Expectation step**

As mentioned earlier, the hidden variables are the states reached by the process while generating the observed sequence. Let us introduce useful auxiliary variables deriving from the hidden variables:

- $N(q, q')$ : the number of times the process goes from state $q$ to state $q'$ in $\mathcal{H}$,
- $V(q, \mathtt{a})$ : the number of times the process reaches state $q$ and emits $\mathtt{a}$ in $\mathcal{H}$,
- $S(q)$ : the number of times the process starts in state $q$.

**Conditional expectation of** $N(q, q')$: The variable $N(q, q')$ can be decomposed as a sum of indicator variables : $N(q, q') = \sum_{t=0}^{l-1} \mathbb{1}\{X_{t+1} = q', X_t = q\}$. Therefore the expectation of $N(q, q')$, denoted by $\overline{N}(q, q')$, can be computed as follows:

$$\overline{N}(q, q') = \mathbb{E}\left[N(q, q') \mid s\right] = \mathbb{E}\left[\sum_{t=0}^{l-1} \mathbb{1}\{X_{t+1} = q', X_t = q\} \,\middle|\, s\right] \quad (2.24)$$

$$= \sum_{t=0}^{l-1} P[X_{t+1} = q', X_t = q \mid s] \quad (2.25)$$

The probability $P[X_{t+1} = q', X_t = q \mid s]$ can be readily obtained using the forward and backward variables:

$$P[X_{t+1} = q', X_t = q \mid s] = \frac{\alpha(q,t)\ A(q,q')\ B(q', s_{t+1})\ \beta(q', t+1)}{\sum_{q,q' \in Q} \alpha(q,t)\ A(q,q')\ B(q', s_{t+1})\ \beta(q', t+1)} \quad (2.26)$$

The numerator computes the probability to generate $s_{:t}$ while reaching state $q$ then to perform the transition $q \to q'$ and to emit $s_{t+1}$ and finally to generate $s_{t+1:}$ from $q'$. The denominator normalizes this expression in order to get a proper probability measure. The expected number of times the process transits from state $q$ is $\overline{N}(q, .) = \sum_{q' \in Q} \overline{N}(q, q')$.

**Conditional expectation of** $V(q, \mathtt{a})$: The variable $V(q, \mathtt{a})$ can be decomposed as $\sum_{t=0}^{l} \mathbb{1}\{X_t = q, O_t = \mathtt{a}\}$. The expectation of $V(q, \mathtt{a})$, denoted by $\overline{V}(q, \mathtt{a})$, is obtained as follows:

$$\overline{V}(q, \mathtt{a}) = \mathbb{E}\left[V(q, \mathtt{a}) \mid s\right] = \sum_{t=0}^{l} P[X_t = q, O_t = \mathtt{a} \mid s] \quad (2.27)$$

$$= \frac{\sum_{\{t \mid s_t = \mathtt{a}\}} \alpha(q,t)\beta(q,t)}{\sum_{t=0}^{l} \alpha(q,t)\beta(q,t)} \quad (2.28)$$

The expected number of times the process reaches state $q$ is $\overline{V}(q,.) = \sum_{\mathtt{a}\in\Sigma} \overline{V}(q,\mathtt{a})$. It should be noted that the difference between $\overline{N}(q,.)$ and $\overline{V}(q,.)$ is that $\overline{N}(q,.)$ does not take a possible occurrence of $q$ as the last state of a path into account.

**Conditional expectation of** $S(q)$: the variable $S(q)$ can be restated as $S(q) = \mathbb{1}\{X_0 = q\}$. The expectation of $S(q)$, denoted by $\overline{S}(q)$, is computed as follows:

$$\overline{S}(q) = P[X_0 = q \mid s] = \frac{\iota(q)B(q,s_0)\beta(q,0)}{\sum_{q\in Q} \iota(q)B(q,s_0)\beta(q,0)} \tag{2.29}$$

**Maximization step**

Once the expected values of the auxiliary variables have been computed, the maximum likelihood estimators of the model parameters are

$$A(q,q') = \frac{\overline{N}(q,q')}{\sum_{q'\in Q} \overline{N}(q,q')}, \quad B(q,\mathtt{a}) = \frac{\overline{V}(q,\mathtt{a})}{\sum_{\mathtt{a}\in\Sigma} \overline{V}(q,\mathtt{a})} \quad \text{and} \quad \iota(q) = \overline{S}(q) \tag{2.30}$$

for all $q, q' \in Q$ and for all $\mathtt{a} \in \Sigma$. The Expectation and Maximization steps are iterated until convergence of the likelihood function.

Unlike for the likelihood computation, both recurrences, forward and backward, are required in the Baum-Welch algorithm. Indeed, computing the expected number of times a transition $q \to q'$ is used while generating a sequence $s$ requires to know the probabilities to generate $s_{:t}$ while reaching $q$ (forward recurrence) and to generate $s_{t:}$ while leaving from $q'$ (backward recurrence), for each time index $0 \leq t < l$. For this reason, the Baum-Welch algorithm is also known as the forward-backward algorithm. The time complexity is $\Theta(|s|\ n_t)$ per iteration where $n_t$ is the number of transitions in the model. The Baum-Welch algorithm is presented here for a single training sequence $s$. It can be readily adapted in order to deal with a sample of training sequences (see [107]). Furthermore, improvements have been proposed in order to face numerical issues due to the numerous products of probabilities in the forward and backward recursions (see [107] and erratum [126]).

# Chapter 3

# Kernel methods

*For the last decade, kernel methods have become one of the most important and challenging fields of research in the machine learning community. In a few words, kernel methods denote learning schemes making solely use of the data through dot products between instances. The* kernel trick *aims at substituting the classical dot product by a function called a* kernel. *Doing so allows one to apply the considered learning method in a suitable space, called the* feature space, *induced by the kernel. For instance, linear methods such as the perceptron can be performed in a possibly non-linear feature space. Beyond non-linear data mapping, the use of kernels enables one to deal with structured data which are not lying in a vector space, e.g. strings, trees or graphs. Kernel methods are used in this thesis in order to solve sequence classification problems (see part III). Several kernels for strings are reviewed in chapter 6. In chapter 7, we propose an original kernel based on First Passage Times (FPT) between occurrences of substrings. Support Vector Machines (SVM) are one of the most important kernel methods and are closely related to the statistical learning theory introduced by Vapnik [130]. Basically, a SVM is a binary classifier computing the maximal margin hyperplane in the space induced by a kernel. There are extensions of SVM to handle regression, ranking and outlier detection problems. Learning a SVM from data amounts to solve a convex optimization problem. Therefore, one is guaranteed to find the global optimum of the objective function, i.e. the hyperplane having the largest margin. Another advantage is that the number of parameters to learn is linear in the number of training instances and independent of the dimension of the input space. This makes the method suitable to handle high-dimensional data. This chapter is organized as follows. Section 3.1 reviews the basis of the statistical learning theory which gives theoretical foundations to the generalization ability of large margin methods. Section 3.2 provides definitions and properties about kernel functions and positive definite matrices. Finally, section 3.3 introduces the SVM*

*for binary classification problem. Connection with the statistical learning theory are highlighted in subsection 3.3.4. Practical implementation issues are briefly reviewed in subsection 3.3.5.*

## 3.1 Statistical learning theory

In this section, we review the basis of the statistical theory introduced by Vapnik [130] . We focus on binary classification problems where we have a training set consisting of $n$ learning instances (also called learning examples) $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$ where $\boldsymbol{x}_i$ is an instance living in some input space $\mathcal{X}$ and $y_i \in \mathcal{Y} = \{-1, +1\}$ denotes its class or its label. The data are assumed to be independently and identically drawn (i.i.d.) from some unknown but fixed probability distribution $P[X, Y]$. The objective is to select a decision function (also called a *classifier*) $f : \mathcal{X} \to \mathcal{Y}$ from a class of functions $\mathcal{F}$ in order to predict the label of new instances.

### 3.1.1 The functional and empirical risks

In order to measure the performance of a classifier, one classically uses a loss function $Err : \mathcal{X} \times \mathcal{Y} \times \mathcal{Y} \to [0, \infty)$ such as the zero-one loss function defined as follows:

$$Err_{0/1}(\boldsymbol{x}, y, f(\boldsymbol{x})) = \frac{1}{2}|f(\boldsymbol{x}) - y| \tag{3.1}$$

This function returns 0 if the prediction $f(\boldsymbol{x})$ and the true label $y$ agree and 1 otherwise. Computing the expectation of this loss function over the joint distribution $P[\boldsymbol{x}, y]$ defines the *functional risk*:

$$R[f] = \int_{\mathcal{X}\times\mathcal{Y}} \frac{1}{2}|f(\boldsymbol{x}) - y| \, dP[\boldsymbol{x}, y] \tag{3.2}$$

In other words, the risk corresponds to the probability of misclassification. The optimal decision function $f^{opt}$ is the one with the lowest risk.

$$f^{opt} = \arg\min_{f\in\mathcal{F}}(R[f]) \tag{3.3}$$

Unfortunately, the minimization of the functional risk requires to have the complete data distribution $P[\boldsymbol{x}, y]$ which is not available in practice. Instead, one can compute the average of the loss function over training data, which is called the *empirical risk*:

$$R_{emp}[f] = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2}|f(\boldsymbol{x}_i) - y_i| \tag{3.4}$$

One of the major results of statistical learning theory states that it is meaningless to select $f \in \mathcal{F}$ as the minimizer of the empirical risk without adequately restricting the class of functions $\mathcal{F}$. Intuitively, if $f$ can be any function $\mathcal{X} \to \mathcal{Y}$, a perfect mapping of the training examples to their label behaving randomly over other instances, would have an empirical risk equal to zero. However, the obtained classifier has not learned anything but the training data. Its functional risk, *i.e.* its expected prediction error rate, will be large. On the other hand, if $\mathcal{F}$ is too poor, the classifier might approximate inaccurately the true mapping to be learned, leading to large empirical and functional risks. This is classically known as the *bias-variance* or *overgeneralization-overfitting* trade-off.

### 3.1.2 Capacity, VC dimension and VC bounds

The Vapnik-Chervonenkis (VC) theory allows one to bound the functional risk while knowing only the empirical risk and the *capacity* of the function class $\mathcal{F}$. Intuitively, the capacity of a class of functions measures the complexity of the boundaries it defines or, in other words, its ability to separate a set of points into two regions[1]. Classical capacity functions are the *shattering number* and the *VC dimension*.

**Definition 29 (Shattering number)** *The* shattering number *of a set of functions, denoted by $\mathcal{N}(\mathcal{F}, n)$, is the number of ways the functions in $\mathcal{F}$ can separate a set of $n$ points into two classes.*

The maximal number of ways a function class $\mathcal{F}$ can separate a set of $n$ points is $2^n$. In this case, we say that the set of functions $\mathcal{F}$ *shatters* the $n$ points. The maximal number of points $h_{VC}$ that can be shattered by a function class $\mathcal{F}$ is called the VC dimension of $\mathcal{F}$.

**Definition 30 (VC dimension)** *The VC dimension of a function class $\mathcal{F}$, denoted by $h_{VC}$, is the maximal number of points that can be shattered by $\mathcal{F}$.*

To illustrate this point, let us consider the VC dimension of the class of linear functions in $\mathbb{R}^2$. One can observe in Figure 3.1 that linear functions in $\mathbb{R}^2$ can shatter a set of three points, eventough it cannot shatter any set of three points. In contrast, there is no set of 4 points in $\mathbb{R}^2$ that can be shattered by this class of functions. Consequently, the VC dimension of linear functions in $\mathbb{R}^2$ is 3. More generally, the following proposition applies to hyperplanes defined in $\mathbb{R}^n$.

[1]The capacity presented here is related to classification problems. Extensions of this notion also apply to the regression setting.

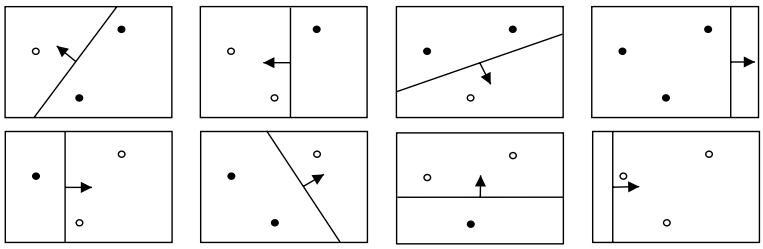


Figure 3.1: A set of 3 points which is shattered by the class of linear functions in $\mathbb{R}^2$. Filled points are classified as positive (*i.e.* +1) while non-filled points are classified as negative (*i.e.* -1).

**Proposition 4** *The VC dimension of hyperplanes in* $\mathbb{R}^n$ *is* $n+1$.
**Proof.** *see [19].*

This proposition is important for the Support Vector Machines (SVM) presented in section 3.3 since it is a linear classifier performed in a possibly high-dimensional feature space. We will see that the high VC dimension of such hyperplanes can be controlled by maximizing the margin between the two classes.

Using the VC dimension, one can derive a confidence interval for the functional risk which holds with a probability $1-\delta$:

$$R[f] \leq R_{emp}[f] + \sqrt{\frac{1}{n}\left(h_{VC}\left(ln\frac{2n}{h_{VC}}+1\right)+ln\frac{4}{\delta}\right)} \tag{3.5}$$

It is important to note that the VC bound (3.5) does not rely on the data distribution $P[\boldsymbol{x}, y]$. However, this bound is not tight and therefore, minimizing the bound does not necessarily minimize the true functional risk. Tighter bounds such as the span bound [131] rely on the geometrical configuration of the training points. The term under the square root is called the *confidence term*. Letting the number of training points tend to infinity makes the confidence term vanish and therefore the functional risk is only bounded by the empirical risk. For a fixed amount of training points, the confidence term can be decreased by considering a function class $\mathcal{F}$ with a low VC dimension $h_{VC}$. However, choosing a poor class $\mathcal{F}$ may lead to a large empirical risk $R_{emp}[f]$. Therefore, to minimize the bound (3.5), one has to pick a function having a low empirical risk, from a function class with a reasonable capacity.

### 3.1.3 Structural risk minimization and regularized risk

Model selection, *i.e.* the selection of the hyperparameters[2] of a learning algorithm, can be done using VC bounds by performing the so-called *Structural Risk Minimization* (SRM). First, let us note that in the bound (3.5), the empirical risk is related to a single function while the confidence term is related to a function class. The core of SRM is to define nested function classes $\mathcal{F}_i$ of increasing VC dimensions. For each function class, the empirical risk is minimized. The optimal model is chosen as the one which minimizes the considered VC bound.

The optimization theory and statistics [14] propose an alternative point of view to this kind of problems, which is known as *regularization*. The main idea is to add an extra term to the objective function in order to penalize complex or non-smooth solutions. In our context, this amounts to minimize the empirical risk regularized by a penalty term $p(f)$ preventing to use a function from class with a too high capacity. This objective function is called the regularized risk:

$$R_{reg}[f] = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}|f(\boldsymbol{x}_i) - y_i| + D\,p(f) \tag{3.6}$$

The constant $D$ allows one to tune the importance of the empirical risk versus the complexity of the solution. In section 3.3.3, it will be shown that SVM actually optimizes a relaxed regularized risk.

## 3.2 Kernels and positive definite matrices

In this section, we review fundamental definitions and properties about kernel functions. Basically, a kernel is a function $\mathcal{X} \times \mathcal{X} \to \mathbb{R}$ which measures the similarity between two instances. Kernels can be thought of as a direct generalization of the dot product between two vectors in the Euclidean space. More precisely, a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a kernel if it corresponds to the dot product in some Hilbert space[3] $\mathcal{G}$:

$$k(\boldsymbol{x}, \boldsymbol{z}) = \langle \Phi(\boldsymbol{x}), \Phi(\boldsymbol{z}) \rangle$$

[2] A hyperparameter is a parameter which is not learned/estimated from the training data. In the case of the Support Vector Machines (SVM, see section 3.3), the hyperparameters are the kernel function $k(.,.)$ and additionally the regularization constant $C$ for soft-margin SVM (see section 3.3.3).

[3] A Hilbert space is a space endowed with a dot product and with the norm deriving from this dot product.

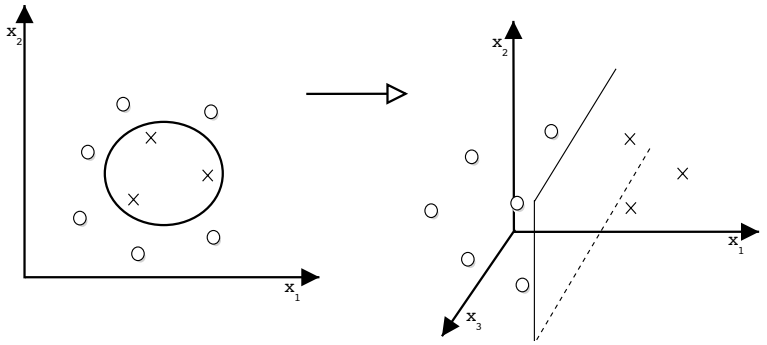



Figure 3.2: A mapping function $\Phi(.)$ making the instances linearly separable.

where $\Phi(.)$ is a function $\mathcal{X} \rightarrow \mathcal{G}$ such that $\Phi(\boldsymbol{x})$ maps the instance $\boldsymbol{x}$ into a Hilbert space $\mathcal{G}$ also called the feature space. Learning algorithms making solely use of the data through dot products between instances can be kernelized by replacing the dot product between instances by a kernel function. The consequence of this substitution is that the learning method is no longer carried out in the input space $\mathcal{X}$ but in the feature space $\mathcal{G}$. Figure 3.2 illustrates the benefits of such a mapping. In the input space $\mathcal{X} = \mathbb{R}^2$, the instances cannot be separated by a linear function while in the feature space $\mathcal{G} = \mathbb{R}^3$ they can be separated by a plane. This plane however correspond to an ellipsoidal function in the input space. The kernelization also known as the *kernel trick* allows one to perform a linear method in a possibly non-linear mapping or embedding of the data. A SVM computes the hyperplane with the largest margin (see definitions 33 and 34) in the feature space induced by a kernel. In general, the number of dimensions of the feature space $\mathcal{G}$ can be very high or even unbounded. Consequently, computing the dot product in such a space can be intractable in practice. Fortunately, in many cases, one can compute the dot product in the feature space via some efficient closed form without mapping explicitly the data points into the feature space.

Although kernel functions are relatively new in the field of machine learning, they were initially studied by mathematicians almost one century earlier. A necessary and sufficient condition for a function to be a valid kernel has been provided by Mercer [91]. In order to state this result, let us introduce the notions of Gram matrix and positive definite matrix. The Gram or the kernel matrix is an $n \times n$ matrix $K$ such that $k_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, *i.e.* the matrix containing the kernel evaluation for any pair of training instances. Positive definite matrices are introduced hereafter.

**Definition 31** *A symmetric matrix* $K \in \mathbb{R}^{n \times n}$ *such that, for all vector* $\boldsymbol{v} \in \mathbb{R}^n$

$$\boldsymbol{v}^T K \boldsymbol{v} \geq 0 \tag{3.7}$$

*is called a* positive definite (PD) matrix[4].

Equivalently, a symmetric matrix $K$ is PD if and only if all its eigenvalues are positive. A PD matrix $K$ is a kind of "square" since it can be decomposed as $K = (U\Lambda^{1/2})(U\Lambda^{1/2})^T$ where $\Lambda^{1/2}$ is a diagonal matrix containing the square root of the eigenvalues of $K$ and $U$ is an orthogonal matrix containing the associated eigenvectors of $K$ stored in columns. Furthermore, PD matrices form a *convex cone*: if $K_1$ and $K_2$ are two PD matrices of the same order then $aK_1 + bK_2$ is also PD for all $a, b \geq 0$. The Mercer condition is stated hereafter.

**Theorem 5 (Mercer's condition)** *The function* $k(.,.) : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ *is a kernel if and only if for all* $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathcal{X}$

$$K = (k(\boldsymbol{x}_i, \boldsymbol{x}_j))_{i,j=1,\ldots,n}$$

*is a positive definite matrix.*
**Proof.** *see [93].*

If a function $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ satisfies the Mercer condition, then it corresponds to a dot product function into some Hilbert space $\mathcal{G}$. Consequently, it enjoys the three fundamental properties of the classical dot product:

- **Positivity** : $k(\boldsymbol{x}_i, \boldsymbol{x}_i) \geq 0$
- **Symmetry** : $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = k(\boldsymbol{x}_j, \boldsymbol{x}_i)$
- **Cauchy-Schwartz** : $k(\boldsymbol{x}_i, \boldsymbol{x}_j)^2 \leq k(\boldsymbol{x}_i, \boldsymbol{x}_i)\ k(\boldsymbol{x}_j, \boldsymbol{x}_j)$

The distance between two instances $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ in the feature space induced by a kernel $k(.,.)$ can be computed as follows:

$$\|\Phi(\boldsymbol{x}_i) - \Phi(\boldsymbol{x}_j)\| = \sqrt{k(\boldsymbol{x}_i, \boldsymbol{x}_i) - 2k(\boldsymbol{x}_i, \boldsymbol{x}_j) + k(\boldsymbol{x}_j, \boldsymbol{x}_j)}$$

The Mercer condition does not provide any information about the underlying kernel mapping or feature space. In fact, the mapping is not unique. A valid mapping can be constructed by using the notion of *Reproducing Kernel Hilbert Space* (for more information, see [115]).

Classical kernels on vectorial data are reviewed hereafter.

[4]For the sake of conciseness, a positive/negative definite matrix actually denotes a semi-positive/semi-negative definite matrix in this thesis.

- **Linear kernel**: $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$
- **Polynomial kernel**: $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle + c)^d$ with $c \geq 0$ and $d \in \mathbb{N}$
- **Gaussian kernel**: $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = exp(\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2/\sigma^2)$ with $\sigma \neq 0$
- **Sigmoid kernel**: $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \tanh(a\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle + c)$ with $a > 0$ and $c < 0$

The linear kernel simply corresponds to the dot product in the input space. Hence, the mapping function $\Phi(.)$ is the identity function. The polynomial kernel with $c = 0$ maps the data points to the space of all monomial of degree $d$. For instance, if the input space is $\mathcal{X} = \mathbb{R}^2$ with axes ($x_1$,$x_2$) then the axes of an induced feature space are $(x_1^2, \sqrt{2}x_1x_2, x_2^2)$. Using $c = 1$ maps the data to the space of all monomials *up to* degree $d$. The feature space induced by the Gaussian kernel is somewhat particular as it is an infinite dimensional space. In fact, the Gaussian kernel maps an instance to a Gaussian function representing the similarity of the considered instance to all possible instances in $\mathcal{X}$. The $\sigma$ parameter allows one to tune the width of the Gaussian function. Using a large $\sigma$ makes an instance similar to many other instances (even if they are far away) while letting $\sigma$ tending to zero makes an instance only similar to itself. In the latter case, the Gram matrix tends to the identity matrix. It should also be noted that the instances are on a unit ball (since $k(\boldsymbol{x}_i, \boldsymbol{x}_i) = 1$ for all $\boldsymbol{x}_i \in \mathcal{X}$) and are located in the same orthant of the feature space (since $k(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq 0$ for all $\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathcal{X}$). The use of the sigmoid kernel allows one to implement a Multiple Layer Perceptron (MLP) having one hidden layer, with a SVM. The number of hidden units in the hidden layer is automatically chosen as it corresponds to the number of so-called *support vectors* (see section 3.3.2). However, the SVM objective function does not correspond to the squared error function used traditionally with MLP. Unlike the other kernels presented here, the sigmoid kernel is actually not (semi-)positive definite but good results are obtained in practice [115].

New kernels can be composed from existing kernels satisfying the Mercer condition. Let $k_1(.,.)$ and $k_2(.,.)$ denotes two Mercer kernels, $a \in \mathbb{R}_0^+$ and $K$ a positive definite matrix, then the following functions are also valid kernels:

- $k(\boldsymbol{x}, \boldsymbol{z}) = k_1(\boldsymbol{x}, \boldsymbol{z}) + k_2(\boldsymbol{x}, \boldsymbol{z})$
- $k(\boldsymbol{x}, \boldsymbol{z}) = ak_1(\boldsymbol{x}, \boldsymbol{z})$
- $k(\boldsymbol{x}, \boldsymbol{z}) = k_1(\boldsymbol{x}, \boldsymbol{z})k_2(\boldsymbol{x}, \boldsymbol{z})$
- $k(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{x}^T K \boldsymbol{z}$

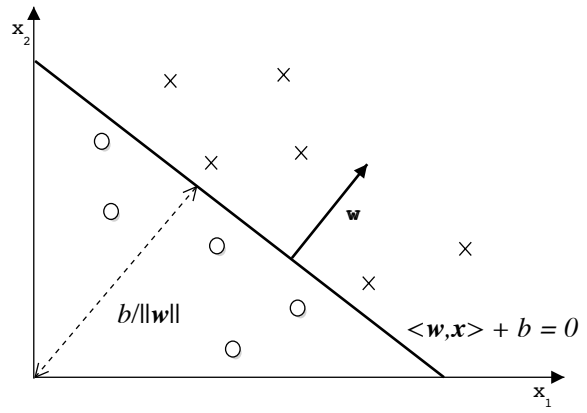



Figure 3.3: A linear classifier $g(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b$ in the input space $\mathcal{X} = \mathbb{R}^2$.

These compositions directly derive from the properties of PD matrices (*i.e.* positive eigenvalues and convex cone) discussed above.

Beyond non-linear mapping, the use of kernels allows one to handle non-vectorial data such as graphs, trees and sequences [118]. Chapter 6 reviews classical kernel on strings whereas chapter 7 introduces a new string kernel based on First Passage Times (FPT) features. These kernel are used with SVM in order to perform discrete sequence classification.

## 3.3 Support Vector Machines (SVM)

### 3.3.1 Definitions

This section reviews SVM in the context of supervised binary classification. A SVM implements a linear function $g : \mathcal{X} \to \mathbb{R}$ :

$$g(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b \tag{3.8}$$

where $\boldsymbol{w} \in \mathcal{X}$ and $b \in \mathbb{R}$ are the parameters of the linear function or hyperplane. An instance $\boldsymbol{x}$ is classified according to the sign of $g(.)$, *i.e.* the decision function is $f(\boldsymbol{x}) = \text{sgn}(g(\boldsymbol{x}))$. This point is illustrated for $\mathcal{X} = \mathbb{R}^2$ in Figure 3.3. The $\boldsymbol{w}$ vector defines the slope of the hyperplane while the bias $b$ translates the hyperplane from the origin by an Euclidean distance equal to $b/\|\boldsymbol{w}\|$.

The notion of *margin* is one of the core components of SVM and is closely related to Vapnik's statistical theory. Before formally introduce the definition of the margin, let us define the separability of a set of instances.

**Definition 32 (separability of a set of instances)** *A set of $n$ instances $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)$ is* separable *if*

$$\exists\, \boldsymbol{w} \in \mathcal{X}, b \in \mathbb{R} : y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) \geq 0 \quad \forall\, i = 1, \ldots, n$$

That is, a set of points is separable in the input space $\mathcal{X}$ if there exists a hyperplane such that the two half-spaces it defines only contain instances from the same class. There are many algorithms to compute such a hyperplane, the most famous being the Perceptron algorithm [110]. To make things easier, we will first assume that our training data are separable. The non-separable case is addressed in section 3.3.3. Let us now introduce the *functional margin* of an instance with respect to an hyperplane.

**Definition 33 (functional margin of a single instance)** *The functional margin of an instance $(\boldsymbol{x}_i, y_i)$ with respect to the hyperplane $\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b = 0$ is*

$$y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b)$$

The *geometrical margin* of an instance $\boldsymbol{x}_i$ is obtained by dividing its functional margin by $\|\boldsymbol{w}\|$. It corresponds to the signed[5] Euclidean distance from $\boldsymbol{x}_i$ to the closest point of the hyperplane. In the sequel, we will simply use the term "margin" to denote the geometrical margin. The margin of a hyperplane with respect to a set of instances is defined as follows:

**Definition 34 (geometrical margin of a hyperplane)** *The geometrical margin of a hyperplane $\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b = 0$ with respect to a set of $n$ instances $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)$ is*

$$\min_{i=1,\ldots,n} \frac{y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b)}{\|\boldsymbol{w}\|}$$

That is, the smallest margin over all the instances in the set. Given a training set $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)$, a SVM computes the hyperplane that has the largest margin. This hyperplane actually bisects the smallest segment joining the convex hulls of the two classes. This situation is depicted in Figure 3.4. In order to maximize the margin, it is convenient to impose that the closest instances from the hyperplane have a functional margin equal to

[5] If $\boldsymbol{x}_i$ is wrongly classified, the margin is negative.

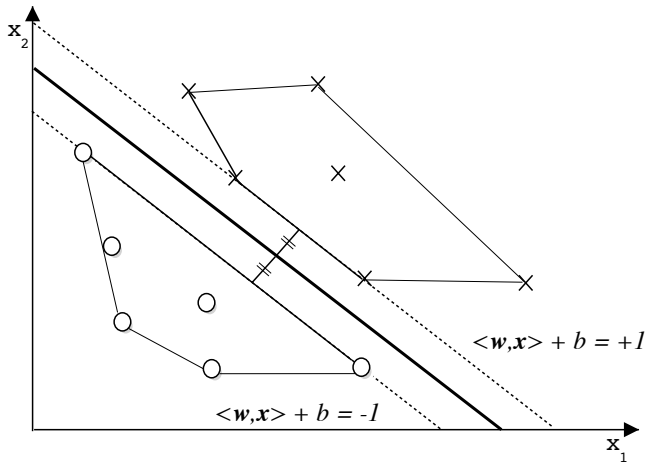



Figure 3.4: The two canonical hyperplanes and the convex hulls of a binary dataset in the input space $\mathcal{X} = \mathbb{R}^2$.

1 in absolute value. That is, the closest positive and negative instances lie respectively on the following *canonical hyperplanes* (see also Figure 3.4):

$$\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b = 1 \tag{3.9}$$
$$\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b = -1 \tag{3.10}$$

The geometrical margin of the hyperplane, in absolute value, is now simply $\frac{1}{\|\boldsymbol{w}\|}$. Hence maximizing the margin can be performed by minimizing $\|\boldsymbol{w}\|$. The relation between the margin maximization and statistical learning theory is discussed in section 3.3.4.

### 3.3.2 Primal and dual formulation

Maximizing the margin of the hyperplane with respect to the training instances can be casted as the following optimization problem:

**QP1** Minimize $W^{\mathrm{pr}}(\boldsymbol{w}, b) = \frac{1}{2}\|\boldsymbol{w}\|^2$

s.t. $y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) \geq 1$

This is a quadratic convex problem where the objective function is the square of the inverse of the margin and where the constraints enforce that the training instances are well classified. The number of variables is the dimension of $\boldsymbol{w}$, *i.e.* the dimension of the input space $\mathcal{X}$, plus one for the bias term.

The dual formulation of **QP1**, presented hereafter, has two main advantages over the primal: (i) the number of variables becomes the number of primal constraints, *i.e.* the number $n$ of instances in the training set and (ii) it allows one to kernelize the algorithm. Let us note that the primal problem **QP1** is not readily kernelizable since the data are not used via dot product of instances. However, it has been recently shown that the primal **QP1** can be reformulated in order to be kernelized [30]. To derive the dual of **QP1**, let us introduce its Lagrangian:

$$L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left[ y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) - 1 \right] \tag{3.11}$$

with $\alpha_i \in \mathbb{R}_0^+$ for all $i = 1, \ldots, n$. The $\alpha_i$ variables are called the *Lagrange multipliers* or the *dual variables*. Basically, one introduces the Lagrangian in order to transform a constrained optimization problem into an unconstrained one. Maximizing the Lagrangian with respect to the dual variables $\boldsymbol{\alpha}$ defines the so-called *barrier* function:

$$h(\boldsymbol{w}, b) = \max_{\boldsymbol{\alpha} \geq 0} L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \begin{cases} W^{\text{pr}}(\boldsymbol{w}, b) & \text{if } y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) \geq 1 \ \ \forall i = 1, \ldots, n \\ \infty & \text{otherwise} \end{cases} \tag{3.12}$$

That is, the barrier function exactly coincides with the objective function of **QP1** whenever the constraints are satisfied. If they are not, the barrier takes an infinite value. This function is called a *barrier* since it prevents the constraints to be violated. Consequently, one can solve the original problem **QP1** by minimizing the unconstrained barrier function $h(\boldsymbol{w}, b)$ :

$$p^{opt} = \min_{(\boldsymbol{w},b)} \max_{\boldsymbol{\alpha} \geq 0} L(\boldsymbol{w}, b, \boldsymbol{\alpha}) \tag{3.13}$$

The notion of *strong duality* essentially amounts to exchange the $\min_{(\boldsymbol{w},b)}$ and the $\max_{\boldsymbol{\alpha} \geq 0}$ in equation (3.13) while preserving the optimal solution:

$$p^{opt} = \min_{(\boldsymbol{w},b)} \max_{\boldsymbol{\alpha} \geq 0} L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \max_{\boldsymbol{\alpha} \geq 0} \min_{(\boldsymbol{w},b)} L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = d^{opt} \tag{3.14}$$

The function $\min_{(\boldsymbol{w},b)} L(\boldsymbol{w}, b, \boldsymbol{\alpha})$ is called the *dual objective function*. The strong duality holds for several classes of convex optimization problems including our case: convex quadratic programming. Optimization techniques such as Interior Point Methods (see section 3.3.5) solve the primal and the dual forms of problems simultaneously. If the problem is only partially solved (*i.e.* the global optimum has not yet been found), the *duality gap* is defined as $p^t - d^t$ where $p^t$ and $d^t$ are respectively the primal and dual solutions at iteration $t$. By strong duality, this non-negative gap vanishes

when the optimal solution is attained. A fundamental result from the duality theory is the Karush-Kuhn-Tucker conditions for optimality. In the case of convex and differentiable problems, it provides sufficient and necessary conditions for a solution to be globally optimal.

**Theorem 6 (KKT for differentiable convex problem [71, 77])** *Let*

**CP1** *Minimize* $W^{\mathrm{pr}}(\boldsymbol{w})$

*s.t.* $\left\{ \; c_i(\boldsymbol{w}) \;\geq\; 0 \quad \forall i = 1, \ldots, n \right.$

*be a continuous optimization problem where* $W^{\mathrm{pr}}(.)$ *and* $c_i(.)$ *for all* $i = 1, \ldots, n$ *are convex and differentiable functions. The Lagrangian is defined as follows:*

$$L(\boldsymbol{w}, \boldsymbol{\alpha}) = W^{\mathrm{pr}}(\boldsymbol{w}) - \sum_{i=1}^{n} \alpha_i c_i(\boldsymbol{w})$$

*where* $\boldsymbol{\alpha}$ *is a* $n$*-dimensional vector such that* $\alpha_i \geq 0$ *for* $i = 1, \ldots, n$*. The solution* $\boldsymbol{w}^{opt}$ *is the minimizer of* **CP1** *if-and-only-if there exists* $\boldsymbol{\alpha}^{opt}$ *such that :*

$$\partial_{\boldsymbol{w}} L(\boldsymbol{w}^{opt}, \boldsymbol{\alpha}^{opt}) = \partial_{\boldsymbol{w}} W^{\mathrm{pr}}(\boldsymbol{w}^{opt}) - \sum_{i=1}^{n} \alpha_i \partial_{\boldsymbol{w}} c_i(\boldsymbol{w}^{opt}) = 0 \quad (3.15)$$

$$\partial_{\alpha_i} L(\boldsymbol{w}^{opt}, \boldsymbol{\alpha}^{opt}) = c_i(\boldsymbol{w}^{opt}) \leq 0 \quad (3.16)$$

$$\alpha_i^{opt} c_i(\boldsymbol{w}^{opt}) = 0 \quad \forall\, i = 1, \ldots, n \quad (3.17)$$

The two first conditions states that the optimal primal-dual pair $(\boldsymbol{w}^{opt}, \boldsymbol{\alpha}^{opt})$ defines a saddle point of the Lagrangian function. The last condition stated in equation (3.17) is called the *complementary* condition. It often provides useful insights about the characteristics of an optimal solution. It states that either the constraint is not active, *i.e.* $c_i(\boldsymbol{w}) < 0$ and the associated dual variable $\alpha_i = 0$, or the constraint is active, *i.e.* $c_i(\boldsymbol{w}) = 0$ and $\alpha_i$ is positive. A physical interpretation of the Lagrange multipliers can be given: it measures how much the solution pushes against the constraints. The KKT conditions for the SVM problem **QP1** are detailed hereafter:

$$\partial_{\boldsymbol{w}} L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \boldsymbol{w} - \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i = 0 \quad (3.18)$$

$$\partial_b L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i y_i = 0 \quad (3.19)$$

$$\partial_{\boldsymbol{\alpha}_i} L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = -y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) + 1 \leq 0 \quad (3.20)$$

$$\boldsymbol{\alpha}_i(y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) - 1) = 0 \quad \forall\, i = 1, \ldots, n \quad (3.21)$$

Interestingly, equation (3.18) allows one to reformulate the primal vector $\boldsymbol{w}$ with dual variables:

$$\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i \tag{3.22}$$

By substituting (3.22) into the Lagrangian defined in equation (3.11), one obtains:

$$\begin{aligned} L(\boldsymbol{w}, b, \boldsymbol{\alpha}) &= \frac{1}{2}\langle \boldsymbol{w}, \boldsymbol{w} \rangle - \sum_{i=1}^{n} \alpha_i \left[ y_i (\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) - 1 \right] \\ &= \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle - \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle + \sum_{i=0}^{n} \alpha_i \\ &= \sum_{i=0}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle \end{aligned} \tag{3.23}$$

It is important to note that the Lagrangian now only depends on dual variables since the remaining bias term $b$ has been canceled out in equation (3.23). Consequently, the inner minimization over $\boldsymbol{w}$ and $b$ in the left part of equation (3.14) has disappeared, leading to the following dual problem:

**QP2** Maximize $W^{\mathrm{dl}}(\boldsymbol{\alpha}) = \sum_{i=0}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$

s.t. $\begin{cases} \sum_{i=1}^{n} \alpha_i y_i = 0 \\ \alpha_i \geq 0 \quad \forall i = 1..n \end{cases}$

This is again a convex quadratic problem but this time with $n$ variables instead of $dim(\mathcal{X}) + 1$ variables, and with $n + 1$ constraints. Each dual variable $\alpha_i$ is associated to a primal constraint, or in other words, to a training instance. As discussed above, a dual variable $\alpha_i$ is strictly positive when the associated primal constraint is active, *i.e.* when the $i$-th instance lies on a canonical hyperplane. These particular instances are called *Support Vectors* (SV). Typically, the number of SV is by far below the number of training instances. The separating hyperplane only depends on SV and the reformulation $\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i$ is often referred to as the *sparse SV expansion* of the hyperplane. Another very important characteristic of **QP2** is that it can directly be kernelized by substituting $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$ by $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

**QP3** Maximize $W^{\mathrm{dl}}(\boldsymbol{\alpha}) = \sum_{i=0}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j k(\boldsymbol{x}_i, \boldsymbol{x}_j)$

s.t. $\begin{cases} \sum_{i=1}^{n} \alpha_i y_i = 0 \\ \alpha_i \geq 0 \quad \forall i = 1..n \end{cases}$

As shown above, the solution of the dual SVM problem does not explicitly provides the hyperplane bias $b$. This bias can however be recovered as follows:

$$b = -\frac{\max_{y_i=-1}(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle) + \max_{y_i=+1}(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle)}{2} \tag{3.24}$$

where $\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle = \sum_{j=1}^{n} \alpha_j y_j k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Finally, the linear function implemented by a SVM is now reformulated as follows:

$$g(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i y_i k(\boldsymbol{x}, \boldsymbol{x}_i) + b \tag{3.25}$$

The decision function used to classify new instances is $f(\boldsymbol{x}) = \text{sgn}(g(\boldsymbol{x}))$.

### 3.3.3 Soft-margin SVM

In section 3.3.1, we made the assumption that our data were linearly separable. In this section, we detail how to compute the maximal margin hyperplane when this assumption does not hold. We distinguish here two reasons why the data may not be linearly separable.

First, the class of functions $\mathcal{F}$ (see section 3.1) implementable by our classifier may be too poor to adequately model the decision function to be learned (also called the target function). For instance, if the target function is quadratic, in general, one will not be able to perfectly separate the instances with a linear function. To circumvent this point, one can use a kernel bringing the instances in some richer feature space (for instance by considering a polynomial kernel with a higher degree or a Gaussian kernel having a tighter width) allowing one to represent the target function linearly.

The second reason why the data may not be separable is the presence of noise. Even tough one is working in a feature space allowing one to represent the target function, the noise on the features or on the labels can prevent from perfectly separate the training instances. A direct solution to face this problem is to work in an even more complex feature space making the data separable. However, doing so amounts to learn the noise present in the data, what can clearly lead to overfitting problems. Furthermore, from the statistical learning viewpoint, using a more complex feature space increases the VC dimension of the classifier. Consequently, the functional risk of the classifier is also likely to increase (see the VC bound presented in equation 3.5).

In order to deal properly with non-separable data, one should allow training instances to be misclassified (*i.e. outliers*) or to lie inside the margin. To do so, one can relax the constraints of **QP1** enforcing the training instance

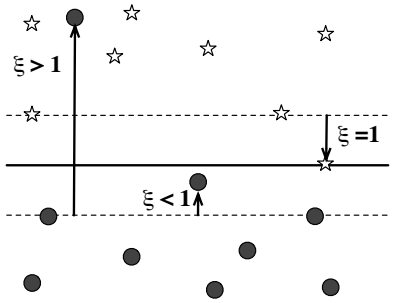



Figure 3.5: Soft-margin SVM and associated slacks

to be correctly classified, by introducing positive *slack variables* $\xi_i$ and a penalization term in the objective function:

**OP1** Minimize $W^{\text{pr}}(\boldsymbol{w}, b, \boldsymbol{\xi}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C.\upsilon(\boldsymbol{\xi})$

$$\text{s.t.} \quad \begin{cases} y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) \geq 1 - \xi_i & i = 1, \ldots, n \\ \xi_i \geq 0 & i = 1, \ldots, n \end{cases}$$

The $\upsilon(\boldsymbol{\xi})$ term introduced in the objective function is used to penalize solutions presenting many training errors. The constant $C$ is called the *regularization* constant[6] It allows one to tune the relative importance of minimizing the training error versus maximizing the margin. For any feasible solution $(\boldsymbol{w}, b, \boldsymbol{\xi})$, misclassified training examples have an associated slack value of at least 1. The situation is illustrated in figure 3.5. Hence, it seems natural to chose a function counting the number of slacks greater or equal to 1 as penalization function $\upsilon(.)$. Indeed, with $C = 1/n$, the penalization term would be equivalent to the empirical risk (see equation 3.4). Unfortunately, the optimization of such a function combined with the margin criterion turns out to be a mixed-integer problem known to be NP-hard [115]. In fact, two approximations of the counting function are commonly used: $\upsilon(\boldsymbol{\xi}) = \sum_{i=1}^{n} \xi_i$ (1-norm) and $\upsilon(\boldsymbol{\xi}) = \sum_{i=1}^{n} \xi_i^2$ (2-norm). These approximations present two peculiarities: 1) The sum of slacks related to examples inside the margin might be considered as errors. 2) Examples with a slack value greater than 1 might contribute more than one error. However, the use of these approximations is computationally attractive as the problem remains convex, quadratic and consequently solvable in polynomial time. In the sequel, we will focus on the 1-norm alternative:

[6]The denomination "regularization constant" for $C$ is somewhat misleading since, strictly speaking, the true regularization constant is actually $D = \frac{1}{2nC}$ and $\|\boldsymbol{w}\|^2$ is the actual regularization term while $\upsilon(\boldsymbol{\xi})$ is a loss term (see equation (3.28)).

**QP4** Minimize $W^{\mathrm{pr}}(\boldsymbol{w}, b, \boldsymbol{\xi}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\xi_i$

$$\text{s.t.} \quad \begin{cases} y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b) \geq 1 - \xi_i & i = 1, \ldots, n \\ \xi_i \geq 0 & i = 1, \ldots, n \end{cases}$$

The next proposition shows that the term $\sum_{i=1}^{n}\xi_i$ is closely related to the empirical risk presented in section 3.1 (see equation 3.4).

**Proposition 5** *Let ($\boldsymbol{w}$,b,$\boldsymbol{\xi}$) be a solution satisfying the constraints of* **QP4**. *The following bound holds for the empirical risk on the training set:*

$$R_{emp}[f] = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}|f(\boldsymbol{x}_i) - y_i| \leq \frac{1}{n}\sum_{i=1}^{n}\xi_i \tag{3.26}$$

This result derives directly from the fact that if an instance $\boldsymbol{x}_i$ is wrongly classified, its associated slack value $\xi_i$ is at least one. The loss function associated to this upper bound is called the *hinge loss* and is defined as

$$Err_{hinge}(\boldsymbol{x}, y, g(\boldsymbol{x})) = \max(0, y.g(\boldsymbol{x})) \tag{3.27}$$

Interestingly, one can make a connection between the optimization of the regularized risk (see equation (3.6)) and the soft-margin SVM. The soft-margin SVM actually optimizes a modified version of the regularized risk, $R^{\mathrm{UB}}_{reg}[f]$, where the empirical risk is upper bounded by $\frac{1}{n}\sum_{i=1}^{n}\xi_i$ and where the regularization term is the square of the inverse of the margin:

$$R^{\mathrm{UB}}_{reg}[f] = \frac{1}{n}\sum_{i=1}^{n}\xi_i + D\|\boldsymbol{w}\|^2 \tag{3.28}$$

where $D = \frac{1}{2nC}$.

Like the hard-margin problem **QP1**, the soft-margin problem **QP4** can be dualized in order to reduce the dimension of the problem to the number of instances and to introduce kernel functions. The dual form of **QP4** is given hereafter:

**QP5** Maximize $W^{\mathrm{dl}}(\boldsymbol{\alpha}) = -\frac{1}{2}\sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) + \sum_{i=0}^{n}\alpha_i$

$$\text{s.t.} \quad \begin{cases} \sum_{i=1}^{n}\alpha_i y_i = 0 \\ 0 \leq \alpha_i \leq C \quad \forall i = 1..n \end{cases}$$

The only difference with **QP3** is that the Lagrange multipliers are now upper bounded by $C$. Three kinds of instances can be distinguished according to their dual variable $\alpha_i$:

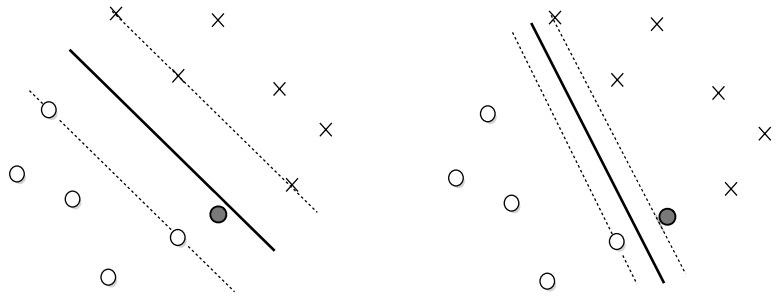


Figure 3.6: Left: A set of instances (the filled circle stands for a test instance) in $\mathbb{R}^2$ with the maximal margin hyperplane computed on training instances. Right: the same set of instances with a hyperplane having a smaller margin.

1. $\alpha_i = 0$ : the instance is correctly classified and does not lie on a canonical hyperplane. This kind of instance is called a non-SV.
2. $0 < \alpha_i < C$ : the instance is well classified and lies on a canonical hyperplane, *i.e.* it is a SV.
3. $\alpha_i = C$ : the example either lies in the margin or is misclassified. It is considered as a SV (as it contributes to the SV expansion of the separating hyperplane) but it is also referred to as an outlier.

### 3.3.4 Relation with statistical learning theory

In this section, we underline connections between the statistical learning theory presented in section 3.1 and SVMs. First, let us consider a simple example providing some intuition about why we should consider hyperplanes with a large margin. The left and the right parts of Figure 3.6 show the same set of instances living in $\mathbb{R}^2$. The filled circle is a test instance, not present while learning the classifier. This set of points is clearly separable. The left part of Figure 3.6 displays the maximal margin hyperplane learned from the training instances. This hyperplane perfectly classifies the training instances as well as the test instance (which falls inside the margin). In contrast, the right part of Figure 3.6 shows a hyperplane perfectly learned on training instances, *i.e.* there is no misclassified training instance, but with a smaller margin. One can observe that this hyperplane wrongly classifies the test instance. To sum up, the margin acts like a "safety" region for the test instances. A larger margin enables safer decisions as the instances can move along a larger radius without being misclassified. Consequently, it offers a

better resistance to noise.

In section 3.1, it was shown that the VC dimension of a hyperplane in $\mathbb{R}^n$ is $n+1$. This result is rather pessimistic, in particular, when we are working in a high-dimensional feature space induced by a kernel (see section 3.2). However, we are not considering any hyperplanes, but those with the largest margin. It is a subclass of linear functions which has its own VC dimension. The following theorem makes the connection between the margin and the VC dimension.

**Theorem 7 (VC dimension for bounded margin hyperplanes [130])** *Let $\mathcal{F}_{\boldsymbol{w}} \triangleq \{\langle \boldsymbol{w}, \boldsymbol{x} \rangle = 0\}$ denotes the class of unbiased hyperplanes such that the closest instances from the hyperplane have a functional margin equal to 1. The VC dimension $h_{VC}$ of the function class $\mathcal{F}_{\boldsymbol{w}}$ with $\|\boldsymbol{w}\| \leq \Lambda$ is bounded by*

$$h_{VC} \leq R^2 \Lambda^2$$

*where $R$ is the radius of the smallest sphere enclosing all the training instances centered at the origin.*

Let us point out the following important remarks about the preceding theorem:

- The bound on the VC dimension, often called the *radius-margin bound*, is inversely proportional to the square of the geometrical margin $1/\|\boldsymbol{w}\|$. Hence, one can control the VC dimension (which itself is related to the generalization ability, see equation 3.5) by maximizing the margin.
- Interestingly, this result is independent from the dimension of the input/feature space where the data live. One can control the VC dimension even in a very high dimensional space, circumventing the curse of dimensionality.
- There are extensions of this theorem including biased hyperplanes and using finer informations about the geometrical configuration of the instances in the space instead of the radius of the smallest enclosing hypersphere [115].

The relation between SVM and structural risk minimization (SRM) is now investigated. SRM optimizes a VC bound (for instance, see equation (3.5)) by minimizing the empirical risk over nested function classes of increasing VC dimensions. Let us restate the function class that can be implemented by a SVM once the kernel $k(.,.)$ and the regularization constant

$C$ have been selected:

$$\mathcal{F}_{k,C} = \left\{ f(\boldsymbol{x}) = \operatorname{sgn}\left( \sum_{i=1}^{n} \alpha_i y_i k(\boldsymbol{x}, \boldsymbol{x}_i) + b \right) \;\middle|\; 0 \leq \alpha_i \leq C,\ i = 1, \ldots, n \right\} \tag{3.29}$$

It consists of linear functions, *i.e.* hyperplanes, defined in the feature space $\mathcal{G}$ induced by the kernel. According to proposition 4, the VC dimension of hyperplanes in $\mathbb{R}^m$ is $m + 1$. This result would prevent one to work in a high-dimensional feature space as the capacity could not be controlled. Fortunately, another result by Vapnik (see theorem 7) directly relates the VC dimension to the margin of hyperplanes. Minimizing (a bound on) the VC dimension can be performed by maximizing the margin. Importantly, this result is independent from the dimension of the feature space.

Inside a function class $\mathcal{F}_{k,C}$, SVM selects a function minimizing an upper bound on the empirical risk (see equation (3.28)) and belonging to a subclass with a controlled VC dimension (by maximizing the margin). In this sense, one can consider that SVM approximates the SRM at the level of the hyperplane parameters. This point is illustrated in Figure 3.7. A complete SRM procedure would also take place at the hyperparameters (*i.e.*, $k(.,.)$ and $C$) level. Unfortunately, such a procedure is difficult to apply in practice for the following reasons: (i) the nested subclasses of increasing VC dimensions are not straightforward to define, especially when considering structurally different kernel functions (*e.g.* the polynomial and the Gaussian kernels) (ii) according to the number of function subclasses, it may become computationally intractable. In practice, the hyperparameters are often chosen using a regular cross-validation procedure. Alternatively, Chapelle and Vapnik [31] have proposed to select the model selection by performing a gradient descent over the radius-margin bound (see theorem 7) with respect to the hyperparameters. In [22], we proposed to perform model selection according to the $\mathcal{F}_\beta$ measure, combining precision and recall, which is better suited in case of unbalanced data. A potential drawback of such techniques is that the radius-margin bound is not convex with respect to the hyperparameters and, therefore, the gradient descent procedure is likely to get trapped into local minima. The regularization constant $C$ can be tuned without having to explore all values in a range by computing the *regularization path* [60]. It allows one to train a SVM for all possible values of $C$ at a similar computational cost than a standard SVM. Another approach [79] attempts to learn the hyperparameters at the training time using Semidefinite Programming (SDP). For instance, it can learn the weights of a linear mixture of kernels as well as the regularization constant $C$ when the $l_2$ norm is considered for the slack variables (see section 3.3.3). To the best of our knowledge, this

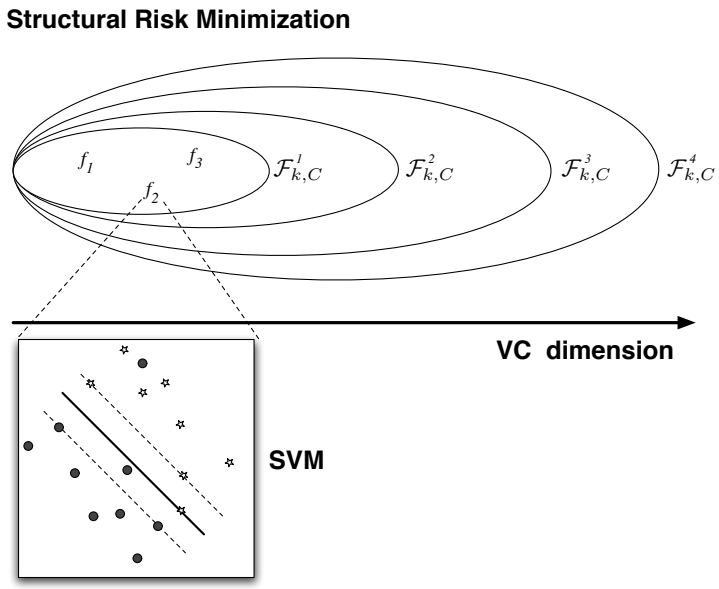



Figure 3.7: SVM in the context of the structural risk minimization. Nested function classes $\mathcal{F}^i_{k,C}$ of increasing VC dimensions are defined according to the hyperparameters, *i.e.* the kernel $k(.,.)$ and the regularization constant $C$. For fixed hyperparameters (*i.e.* inside one of these function classes), SVM computes a function minimizing an upper bound on the empirical risk and belonging to a subclass with a controlled VC dimension (by maximizing the margin).

technique only works in a transductive setting, *i.e.* when the test instances are known (without their labels) at training time.

### 3.3.5 Implementation

Convex quadratic problems such as **QP2** and **QP5** are well-known in the continuous optimization community [14]. They can be solved using off-the-shelf optimization packages such as LOQO [129], SeduMi [122] or MOSEK [4]. These softwares rely on Interior Points Methods (IPM) which simultaneously solve the primal and dual forms of the considered problem. This is done iteratively until the duality gap (*i.e.* the difference between the primal and dual objectives) has vanished, which means that the optimal solution has been found. At each iteration $t$, the primal and dual variables are linearized, *i.e.* $w_i^t = w_i^{t-1} + \Delta w_i$ and $\alpha_i^t = \alpha_i^{t-1} + \Delta\alpha_i$, and a relaxed[7] KKT

[7]The complementary KKT conditions are relaxed such that the product of the primal constraints with their respective Lagrange multiplier needs not to be null but below a

system (see section 3.3.2) is solved for the update variables $\Delta w_i$ and $\Delta\alpha_i$. The primal-dual pairs $(\boldsymbol{w}^t, \boldsymbol{\alpha}^t)$ obtained at each iteration $t$ define a path of feasible solutions which is called the *central path*. The relaxed KKT system can be solved using a *predictor-corrector* strategy. In a few words, the predictor step solves the relaxed KKT system while ignoring the quadratic terms (*i.e.* products of update variables). Then, the corrector step substitutes the obtained values into the quadratic terms (which are now considered as constant) and resolves the system for the updates $\Delta w_i$ and $\Delta\alpha_i$. This technique is simple to implement (no derivatives of the functions are required) and provides accurate results whenever the updates are small. The linear system involved in the predictor and corrector steps can be efficiently solved using numerical routines such as the Cholesky decomposition [104], in a time complexity $\mathcal{O}(n^3)$ where $n$ denotes the number of dual variables or training instances. More details about IPM can be found in [129] and a specialized code for solving the SVM classification problems is proposed in [20].

Even though IPM are efficient to solve quadratic problems, the cubic time complexity required at each iteration does not allow one to solve problems having more than a few thousands variables (*i.e.* training instances). However, in machine learning, one has often to face larger datasets such as, for instance, the MNIST dataset for optical character recognition which contains 60,000 training instances.

In order to improve the training efficiency, one can take benefit from the particular form of the SVM problems. In many cases the solutions of **QP2** and **QP5** are very sparse, *i.e.* there are many dual variables $\alpha_i = 0$, since the number of support vectors is generally by far below the total number of training instances. Therefore, a significant speed-up would be made if one could only optimize the variables which will eventually be non-zero, *i.e.* support vectors. This statement is the main motivation of *chunking* or *decomposition* methods. In such techniques, the set of variables is split into a *working set* containing variables to be optimized, and a *fixed* set containing frozen variables (*i.e.* considered as constants). The *chunking* methods consist in iteratively building working/fixed sets and to solve the related subproblems until the KKT optimality conditions are satisfied. There are several chunking strategies. The first, proposed by Osuna [100], selects the active variables randomly and, after having solved the subproblem, it keeps the variables related to SV in the working set for the next iteration. Using this strategy, the size of the working set tends to grow at each iteration, mak-

prescribed threshold. At each iteration, this threshold is decreased, leading eventually to the original KKT system.

ing the procedure quite slow because of the cubic time complexity of IPM solvers. In contrast, another strategy proposed by Joachims [69] considers a fixed size working set containing typically 10 variables. The active variables are selected according to the gradient of the objective function. This technique, implemented in SVM$^{Light}$, decreases significantly the training time, allowing one to deal with dataset containing about $10^5$ instances.

Fixing the working set size to 2 defines a particular case of chunking called *Sequential Minimal Optimization* (SMO) [102]. In this case, the subproblems can be solved analytically, without requiring a numerical solver. Furthermore, the memory requirements are very low since the maximal memory footprint is about the size of the training set. The working set selection criterion is equivalent to the gradient-based criterion applied to this particular case. In practice, SMO, implemented in libSVM [29], offers a performance almost as good as the chunking strategy proposed by Joachims.

Other implementation improvements include *shrinking* and *kernel caching*. The shrinking amounts to virtually remove variables from the problem. Typically, when a dual variable is associated to a non-SV ($\alpha_i = 0$) or to an outlier ($\alpha_i = C$) for several iterations, the variable is removed from the problem. A potential issue of this technique is that the KKT conditions do not guarantee anymore that the original problem (*i.e.* containing all the variables) is solved. Therefore, when an optimal solution is found, one has to check if the complete KKT conditions are satisfied. In the case they are not, one has to reintroduce the shrinked variables and to restart. The kernel caching is especially important when the evaluation of the kernel function is a computational burden. In chunking methods, when the working set has been selected, the Gram submatrix relative to active instances is required to solve the considered subproblem. Kernel caching amounts to save the kernel evaluations in a fixed memory bunch. Available Gram entries can be reused during next chunking iterations without having to recompute them. When the kernel cache is full, one has to discard kernel entries in order to store the more recent ones. The discarded entries should be selected such as the number of *cache miss*, *i.e.* the number of times a Gram entry must be (re)computed, is minimized. In most of the SVM implementations, the discarded entries are selected according to the Least Recently Used (LRU) criterion. More details about the efficient implementation of SVM can be found in [115] and [20].

# Part II

# Hidden Markov Model induction

# Chapter 4

# An overview of HMM induction techniques

*This chapter reviews some important techniques to the structural induction of Hidden Markov Models (HMMs). We distinguish here three kinds of approaches: (i) methods performing structural induction through parameter estimation (ii) techniques based on state merging or state splitting and (iii) approaches optimizing a discriminative criterion rather than the likelihood or a maximum a posteriori (MAP) criterion.*

*Methods based on parameter estimation apply to a fixed input graph. If no prior knowledge about the model structure is available, a fully connected graph is usually considered. After parameter fitting, useless transitions are trimmed off from the model. For instance, one can apply the Baum-Welch algorithm (see section 2.3.3) on a fully connected graph and trim transitions with a small probability. A more sophisticated technique due to Brand [16] maximizes a MAP criterion with entropic priors enforcing the sparseness of parameters.*

*State merging techniques start with a model overfitting the training sequences (typically containing many states) and successively merge "compatible" states in order to improve the generalization ability of the model. The ALERGIA [26, 27] and MDI [125] algorithms apply to Probabilistic Deterministic Finite state Automata (PDFA) which form a proper subclass of HMMs [45]. These techniques essentially differ on the definition of "compatible" states. The former technique merges states having a close suffix distributions while the second technique merges states leading to a small divergence increase relatively to the model size reduction. Some learnability results about probabilistic automata (and HMMs) are also discussed. The Bayesian state merging due to Stolcke et al. [121] is perhaps the most referenced work in the field of HMM induction. In contrast with ALERGIA and MDI, it applies to the general class of HMMs. It maximizes a MAP criterion*

*with Minimum Description Length (MDL) priors on the structural parameters (e.g. the number of states) and Dirichlet priors on the multinomial distributions related to transition and emission probabilities. State splitting techniques start with a small initial model and iteratively split states in order to improve the fit to the training data. In this context, a technique has been proposed by Ostendorf [99] to learn left-to-right HMMs. First an initial model is estimated with the Baum-Welch algorithm. At each iteration, a state is split into two new states so as to maximize the expected likelihood on a constrained subset of the parameters.*

*All the preceding techniques aim at modeling the generative process of the training sequences. Alternatively, HMMs can be specifically trained to be used in a discriminative setting such as sequence classification or sequence labeling. In this context, we briefly overview techniques optimizing the conditional likelihood and methods relying the maximization of a margin.*

*This chapter is organized as follows: section 4.1 reviews induction techniques through parameter estimation including Baum-Welch applied on fully connected graphs (subsection 4.1.1) as well as the technique proposed by Brand based on entropic priors (subsection 4.1.2). Section 4.2 reviews methods relying on state merging and state splitting. More precisely, subsection 4.2.1 presents the ALERGIA and MDI algorithm, subsection 4.2.2 details the Bayesian state merging algorithm due to Stolcke et al. and subsection 4.2.3 reviews the maximum likelihood state splitting technique proposed by Ostendorf. Section 4.3 presents an overview of some discriminative learning techniques, including methods optimizing the conditional likelihood or a margin-based criterion.*

## 4.1 Structural induction by parameter estimation

This section presents structural induction techniques relying on the estimation of parameters. In such approaches, an initial model with a prescribed topology is estimated and the probabilistic parameters with a small magnitude are trimmed off from the model. Section 4.1.1 presents the Baum-Welch algorithm applied to fully connected graphs while section 4.1.2 details the MAP estimation of parameters with an entropic prior biasing towards parameter extinction. Parameter trimming and extinction are two related but distinct notions. Trimming structurally removes a parameter of small magnitude whereas extinction numerically pushes a probabilistic parameter towards a null probability during the estimation. Consequently, extinction supports the trimming of parameters.

### 4.1.1 Baum-Welch on fully connected graphs

The most direct way of performing HMM induction is to apply the Baum-Welch algorithm (see section 2.3.3) to a fully connected graph. The estimated model remains fully connected since Baum-Welch is an iterative procedure unable to set parameters exactly to zero. Let us recall that Baum-Welch only finds a local maximum of the likelihood function. Abe and Warmuth [1] have shown theoretically that maximizing the sample likelihood globally is not feasible in polynomial time. Let us however point out that this result states that the time complexity is non-polynomial in the size of the alphabet which is often considered as constant in practical problems. In order to effectively learn the model structure with Baum-Welch, one has to structurally remove (*i.e.* set to zero) parameters of small magnitude. Several alternative trimming procedures can be defined. We detail here the implementation, called StructBW, used in the induction experiments presented in section 5.6. The pseudo-code of StructBW is given in Algorithm 1. It takes three input parameters: (i) a training sample $S$, (ii) a set $N_s$ of increasing model sizes to test and (iii) a number $n_r$ of random initializations of the probabilistic parameters. The outer-most loop iterates on the model sizes. First, a fully connected graph having $n_s$ states is obtained using the function `fullMeshed`. The Baum-Welch algorithm is applied on this structure using $n_r$ random initializations for the probabilistic parameters; the parameters leading to the best log-likelihood are kept. The inner loop is concerned with transition trimming. The transition having the smallest probability is trimmed by the function `trimSmallest` and the model is normalized accordingly. The Baum-Welch is applied on the trimmed model with the current parameter values (no random initialization is used here). The obtained model is kept if the sample likelihood is not decreased. The trimming procedure is iterated until a transition removal causes a likelihood decrease. The algorithm returns the estimated models for all considered sizes. The optimal model size is selected as the one offering the largest likelihood while applied to an independent validation set of sequences. Other criteria, not requiring a validation set, can be considered for selecting the optimal model size, the most popular being the Akaike's Criterion (AIC) [2] and the Bayesian Information Criterion (BIC) [116]. Both criteria are instances of the following formula to be maximized:

$$\log(P[S \mid \rho]) - D\,|\rho| \tag{4.1}$$

where $|\rho|$ denotes the number of probabilistic parameters of the model. The AIC is instantiated by setting $D = 1$ while the BIC is obtained by setting $D = \log(|S|)/2$. The first term of formula (4.1) denotes the log-likelihood of the sample $S$ with respect to a model parametrized by $\rho$. The second

**Algorithm** StructBW
**Input**: • A training sample $S$
• A set $N_s$ of model sizes
• The number $n_r$ of restarts for Baum-Welch
**Output**: A collection of HMMs: $H_1, \ldots, H_{|N_s|}$

$i \leftarrow 1$
**for** $n_s \in N_s$ **do**
  $G_i \leftarrow$ `fullMeshed`$(n_s)$
  $H_i \leftarrow$ `BauwWelch`$(G_i, S, n_r)$
  $Lik_i \leftarrow$ `likelihood`$(H_i, S)$
  **repeat**
    $Lik_{last} \leftarrow Lik_i$
    $H_{tmp} \leftarrow$ `trimSmallest`$(H_i)$
    $H_{tmp} \leftarrow$ `BauwWelch`$(H_{tmp}, S)$
    $Lik_{tmp} \leftarrow$ `likelihood`$(H_{tmp}, S)$
    **if** $Lik_{tmp} \geq Lik_i$ **then**
      $H_i \leftarrow H_{tmp}$
      $Lik_i \leftarrow Lik_{tmp}$
  **until** $Lik_{tmp} \geq Lik_{last}$;
  $i \leftarrow i + 1$
**return** $H_1, \ldots, H_{|N_s|}$

Algorithm 1: The Baum-Welch algorithm performed on fully connected graphs with transition trimming.

term of formula (4.1) aims at penalizing complex models, *i.e.* models with numerous parameters. In this sense, these criteria can be thought of as regularization (see section 3.1.3).

In section 5.4.2, we argue that applying Baum-Welch on a fully connected graph is poorly suited in order to model long-term dependencies. This phenomenon is observed in the experiments presented in section 5.6.

### 4.1.2 MAP estimation with entropic priors

Brand [16] proposes a method for estimating parameters and discovering the structure of probabilistic models with hidden states, HMMs being a particular case. To do so, an entropic prior on the parameters of multinomial distributions is proposed. This prior pushes the parameters towards deterministic values (*i.e.* 0 or 1). Consequently, the obtained models are biased towards sparsity. The estimation procedure relies on the EM algorithm where the M-step is replaced by a MAP estimation including the entropic priors. Doing so, weakly supported parameters are iteratively driven to extinction and surviving parameters approach their maximum likelihood estimate. At each step, the posterior probability of the model given the data is monotonically increased. Additionally a trimming procedure is proposed to structurally remove, before convergence, parameters that would asymptotically decay to zero. We review here the building blocks of this approach, *i.e.* the definition of the entropic prior, the MAP estimate of parameters and the trimming procedure.

Let $\mathcal{P}_{\boldsymbol{\theta}}$ denote a multinomial distribution parametrized by $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_k)$. The parameter $\theta_i$ defines the probability that the $i$-th outcomes occurs and $\sum_{i=1}^{k} \theta_i = 1$. The entropic prior for the parameters $\boldsymbol{\theta}$ is defined as follows :

$$P_{ent}[\boldsymbol{\theta}] \propto e^{-H(\boldsymbol{\theta})} = \exp\left(\sum_{i=1}^{k} \theta_i \log \theta_i\right) = \prod_{i=1}^{k} \theta_i^{\theta_i} \tag{4.2}$$

where $H(\boldsymbol{\theta}) = -\sum_{i=1}^{k} P[\theta_i] \log(P[\theta_i])$ denotes the Shannon entropy applied to the multinomial distribution $\mathcal{P}_{\boldsymbol{\theta}}$. The entropic posterior of the model parameters given the observations is provided by

$$P_{ent}[\boldsymbol{\theta} \mid \boldsymbol{c}] \propto P[\boldsymbol{c} \mid \boldsymbol{\theta}] P_{ent}[\boldsymbol{\theta}] = \prod_{i=1}^{k} \theta_i^{\theta_i + c_i} \tag{4.3}$$

where $\boldsymbol{c}$ is a $k$-dimensional vector such that $c_i$ is the number of times the $i$-th outcome was observed. The MAP estimate of the parameters $\boldsymbol{\theta}$ are obtained by maximizing the log of the entropic posterior (defined in equation 4.3) with

respect to $\boldsymbol{\theta}$ and subject to $\sum_{i=1}^{k} \theta_i = 1$. To do so, one defines the following Lagrangian (see section 3.3.2) :

$$L(\boldsymbol{\theta}, \alpha) = \log \left( \prod_{i=1}^{k} \theta_i^{\theta_i + c_i} \right) + \alpha \left( \sum_{i=1}^{k} \theta_i - 1 \right) \tag{4.4}$$

Where $\alpha \in \mathbb{R}^+$ is a Lagrange multiplier enforcing the constraint $\sum_{i=1}^{k} \theta_i = 1$. As this Lagrangian is not convex, the KKT conditions (see theorem 6) only allow one to characterize local optima. Setting the partial derivative of $L(\boldsymbol{\theta}, \alpha)$ with respect to $\theta_i$ to zero yields:

$$\partial_{\theta_i} L(\boldsymbol{\theta}, \alpha) = \frac{c_i}{\theta_i} + \log(\theta_i) + 1 + \alpha = 0 \tag{4.5}$$

Doing so for all the partial derivatives $\partial_{\theta_i} L(\boldsymbol{\theta}, \alpha)$, with $i = 1, \ldots, k$, defines a system of independent transcendental[1] equations. Such a system is solved here using the Lambert function [35] $W(.)$ satisfying the functional mapping $W(y)\exp(W(y)) = y$ which implies that $\log(W(y)) + W(y) = \log(y)$. Defining $y = e^x$ and an additional variable $z \in \mathbb{R}^+$, one obtains

$$\begin{aligned} & -W(e^x) - \log(W(e^x)) + x + \log(z) - \log(z) &=& \ 0 \\ \iff & -W(e^x) + \log(z/W(e^x)) + x - \log(z) &=& \ 0 \end{aligned} \tag{4.6}$$

Defining $x = 1 + \alpha + \log(-c_i)$ and $z = -c_i$, equation (4.6) becomes

$$-W(-c_i e^{1+\alpha}) + \log\left(-c_i / W(-c_i e^{1+\alpha})\right) + 1 + \alpha = 0 \tag{4.7}$$

Consequently, setting

$$\theta_i = \frac{-c_i}{W(-c_i e^{1+\alpha})} \tag{4.8}$$

defines a solution to equation (4.5). Equations (4.5) and (4.8) define a fix point for the Lagrange multiplier $\alpha$. This yields an iterative procedure for jointly computing $\boldsymbol{\theta}$ and $\alpha$. This procedure amounts to repeat the two following steps until convergence: (i) given the current value of $\alpha$ compute $\boldsymbol{\theta}$ using equation (4.8) and (ii) given the new $\boldsymbol{\theta}$ compute $\alpha$ using equation (4.5). In practice, convergence is fast and takes typically 2-5 iterations.

The MAP estimation relying on entropic priors is substituted to the M (maximization) step in the EM algorithm. The vector $\boldsymbol{c}$ of counts does not directly correspond to observations but to the conditional expectations (*i.e.* the expected number of times the parameters are triggered given the data)

[1] A transcendental equation is an equation containing non-polynomial terms.

computed in the E (expectation) step of the EM. Additionally, a procedure is proposed to structurally remove irrelevant parameters. In a few words, a parameter $\theta_i$ can be trimmed whenever the likelihood loss is compensated by a gain in the prior. A simple test, assuming a small value of $\theta_i$, can be computed using the partial derivative of the log-likelihood function: the parameter $\theta_i$ can be trimmed if

$$\theta_i \leq \exp(\partial_{\theta_i} \log(P[S \mid \boldsymbol{\theta}])) \tag{4.9}$$

where $P[S \mid \boldsymbol{\theta}]$ denotes the likelihood of the sample $S$ given the model parameters $\boldsymbol{\theta}$. When applied to HMMs, the MAP parameter estimation with parameter trimming defines a structural induction technique. As for Baum-Welch, the main drawback of this technique is that it only considers models having a fixed number of states. Consequently, one has to perform a complete training for several model sizes and select the optimal size by validation. On the other hand, entropic priors are applied independently to all sets of multinomial parameters in the model while the relations between parameters should be captured using multivariate entropic priors. Finally, a bias towards sparse structures may be inappropriate to model processes with an underlying graph containing dense regions.

Recently, another induction technique [84] using entropic priors has been proposed. This method aims at determining the number of states of left-to-right HMMs. The MAP parameter estimation relies on a Deterministic Annealing (DA) technique rather than on the Lambert function. It also assumes that sequences have been generated along their most probable path, the Viterbi path. One can think of DA as a simulated annealing procedure where the stochastic simulations on the energy surface are replaced by expectations. The algorithm is started with an initial model having a large number of states, an initial temperature and a cooling scheme for the annealing procedure. At each iteration, the parameters are MAP reestimated according to the current temperature and every merging of state pairs decreasing the free energy are applied. In contrast with the approach of Brand [16], this technique is only concerned with finding the correct number of states, consequently no transition trimming is applied. The temperature is then updated according to the cooling scheme. The procedure is iterated until the temperature raises some lower threshold value. Parameters are finally reestimated for a null temperature.

## 4.2 State merging and splitting

This section presents induction techniques relying on states merging/splitting operations. In contrast with methods based on parameter estimation, these

techniques do no apply to fixed size models. Section 4.2.1 presents the ALERGIA and MDI algorithms which induce Probabilistic Deterministic Finite state Automata (PDFA). Section 4.2.2 describes the Bayesian state merging algorithm due to Stolcke which applies to the general class of HMMs. Section 4.2.3 presents the maximum likelihood technique proposed by Ostendorf et al.

### 4.2.1 The ALERGIA and MDI algorithms

The ALERGIA [26, 27] and MDI [125] algorithms are structural induction techniques for learning Probabilistic Deterministic Finite state Automata (PDFA) with final probabilities, from a learning sample. A PDFA is a PNFA $\langle \Sigma, Q, \phi, \iota \rangle$ (see definition 25) such that

$$\forall q \in Q,\ \forall \mathtt{a} \in \Sigma : |\{q' \in Q \mid \phi(q, \mathtt{a}, q') > 0\}| \leq 1 \tag{4.10}$$

That is, the outgoing transitions from each state $q \in Q$ are labeled with distinct symbols. Consequently, one defines $\delta(q, \mathtt{a})$ as the unique successor of state $q$ reached when triggering the transition originated in $q$ and labeled with $\mathtt{a}$. In the sequel the shorthand $\phi(q, \mathtt{a})$ stands for $\phi(q, \mathtt{a}, \delta(q, \mathtt{a}))$. In [26, 27] and [125], PDFA are considered with final probabilities in order to predict the end of sequences. Equivalently, we will consider here PDFA with a dummy final state $q_f$ that can be reached from any state $q \in Q$ using a silent transition $\phi(q, \varepsilon, q_f)$ where $\varepsilon$ denotes the empty sequence. In [45], it is shown that distributions generated by PDFA form a proper subset of distributions defined by PNFA. Since PNFA are equivalent to HMMs (see section 2.2), these induction algorithm do not apply to the general class of HMMs. Both algorithms can be formulated as instances of the generic state merging scheme presented in Algorithm 2. First an initial model,

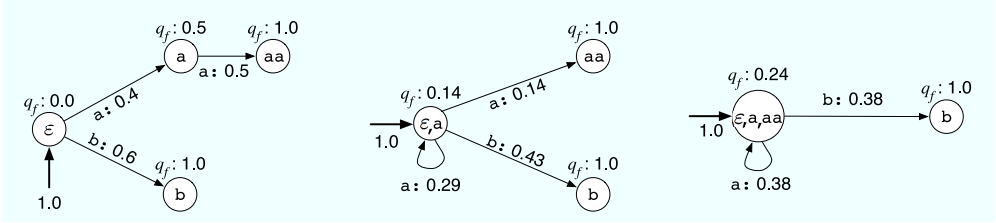
**Input**: • A learning sample $S$
• A precision parameter $\mu$
**Output**: A PDFA $PA$
$PA \leftarrow PPTA(S)$
**while** (There is a compatible candidate pair) **do**
  $(q_1, q_2) \leftarrow$`selectStates(`$PA$`)`
  **if** `compatible(`$q_1, q_2, \mu$`)` **then**
    $PA \leftarrow$`merge(`$PA, q_1, q_2$`)`
**return** $PA$

Algorithm 2: A generic state merging induction algorithm.

| State | $\varepsilon$ | a | b | aa |
|---|---|---|---|---|
| Count | 5 | 2 | 3 | 1 |

| State | $(\varepsilon, \mathtt{a})$ | b | aa |
|---|---|---|---|
| Count | 7 | 3 | 1 |

| State | $(\varepsilon, \mathtt{a}, \mathtt{aa})$ | b |
|---|---|---|
| Count | 8 | 3 |




$PA = PPTA(S)$ $PA = \mathtt{merge}(PA, \varepsilon, \mathtt{a})$ $PA = \mathtt{merge}(PA, (\varepsilon, \mathtt{a}), \mathtt{aa})$

Figure 4.1: Left: the Probabilistic Prefix Tree Automaton (PPTA) built from the sample $S = \{\mathtt{a}, \mathtt{aa}, \mathtt{b}, \mathtt{b}, \mathtt{b}\}$. Center: the **PNFA** resulting from merging states labeled $\varepsilon$ and $a$ in $PPTA(s)$. Right: the **PDFA** resulting from the *determinization by merging*. The counts $C(.)$ relative to the states are displayed above each model.

called a *Probabilistic Prefix Tree Automaton* (PPTA) is constructed from the sample $S$. The PPTA is a tree-shaped automaton representing efficiently the sample's distribution by factorizing common prefixes in $S$. The probability of a sequence $s$ in the sample $S$ is simply the relative frequency of $s$ in $S$. A PPTA constructed from the sample $S = \{\mathtt{a}, \mathtt{aa}, \mathtt{b}, \mathtt{b}, \mathtt{b}\}$ is depicted in Figure 4.1. Probabilities displayed above states stand for the probabilities of transition to the final state $q_f$ which is not explicitly represented here. Each state $q$ is labeled with the unique prefix sequence generated/observed while reaching $q$. To each state $q \in Q$ is associated a counter $C(q) \in \mathbb{N}^0$ providing the number of times state $q$ is attained while generating/observing the sample $S$. These counters are used during merging operations as well as in the compatibility test implemented in the `compatible` function.

At each iteration, two states in the current model are selected for merging. The order in which candidate pairs are probed is defined by the `selectStates` function. The branches of $PPTA(S)$ are first sorted according to the standard order on strings (*i.e.* shorter sequences first then the alphabetical order is applied), as in Figure 4.1. Candidate pairs $(i, j)$, where state $j$ is a predecessor of state $i$, are returned following this order. For instance, the ordered candidate pairs returned by `selectStates` applied to $PPTA(S)$ are $(\mathtt{a}, \varepsilon), (\mathtt{b}, \varepsilon), (\mathtt{aa}, \varepsilon), (\mathtt{aa}, \mathtt{a})$ and $(\mathtt{aa}, \mathtt{b})$. The function `compatible` tests whether two states should be merged according to some statistical criterion and a precision parameter $\mu$. ALERGIA and MDI essentially differ on the definition of this compatibility test. The `compatible` function for both algorithms is detailed later on. Whenever the compati-

bility test is satisfied for a pair $(q_1, q_2)$, the current model is updated by merging $q_1$ and $q_2$ into state $q$ such that

$$\forall q' \in Q, \forall \mathtt{a} \in \Sigma : \phi(q, \mathtt{a}, q') : \frac{C(q_1)\phi(q_1, \mathtt{a}, q') + C(q_2)\phi(q_2, \mathtt{a}, q')}{C(q_1) + C(q_2)} \quad (4.11)$$

If the resulting model is no longer a PDFA (*i.e.* constraint (4.10)) is violated when the resulting structure is non-deterministic), the merging operation is recursively applied to the successors of $q$ until the "determinism constraint" (4.10) is satisfied. This operation is called *determinization by merging*. The center part of Figure 4.1 shows the PNFA resulting from the merging of states $\varepsilon$ and $\mathtt{a}$ in $PPTA(S)$. This model is no longer a PDFA as there are two transitions labeled with $\mathtt{a}$ originated from state $(\varepsilon, \mathtt{a})$. Therefore, the determinization by merging is applied, which consists of merging states $(\varepsilon, \mathtt{a})$ and $\mathtt{aa}$. The resulting model is depicted in the right part of Figure 4.1. The algorithm is iterated until all the candidate pairs in $PPTA(s)$ have been considered. The total number of candidate pairs is in $\mathcal{O}(|PPTA(S)|^2)$ where $|PPTA(S)|$ is the number of states in the PPTA.

In ALERGIA, the `compatible` function implements a test derived from the Hoeffding bound [62]. Formally, two states $q_1$ and $q_2$ are $\mu$-compatible $(0 < \mu \leq 1)$ if the two following conditions hold for all $\mathtt{a} \in \Sigma$:

$$|\phi(q_1, \mathtt{a}) - \phi(q_2, \mathtt{a})| < \sqrt{\frac{1}{2} \ln \frac{2}{\mu}} \left( \frac{1}{\sqrt{C(q_1)}} + \frac{1}{\sqrt{C(q_2)}} \right), \quad (4.12)$$

$$\delta(q_1, a) \text{ and } \delta(q_2, a) \text{ are } \mu-\text{compatible}. \quad (4.13)$$

Condition (4.12) defines the compatibility between each pair of transitions outgoing respectively from state $q_1$ and $q_2$. Condition (4.13) requires the compatibility to be recursively satisfied for every pair of successors of these states[2]. Finally it should be noted that, in the case of small finite samples, state pairs with very low counts can be wrongly considered compatible. To face this issue, Young-Lai and Tompa have introduced some refinements to the compatibility measure [139].

The `compatible` function used in the MDI algorithm [125] is detailed hereafter. MDI aims at inducing PDFA while trading off minimal divergence from the training sample distribution and minimal automaton size. It can be considered as a Bayesian learning method. Indeed a possible solution

[2] If a successor state is undefined for the transition function $\delta$, condition (4.12) can be rewritten for the other successor $q'$ as follows: $\phi(q', \mathtt{a}) < \sqrt{\frac{1}{2} \ln \frac{2}{\mu}} (\frac{1}{\sqrt{C(q')}})$, $\forall a \in \Sigma$.

with null divergence is $PPTA(S)$. It is also a maximum likelihood model built from the learning sample. On the other hand, favoring small automata corresponds to an increased prior probability associated to a reduced automaton size. Trading off both effects is thus equivalent to maximizing the posterior probability of the model given the training data. Let us assume that $PA_0 = PPTA(S)$, $PA_1$ is a temporary solution and $PA_2$ is a tentative new solution that can be derived from $PA_1$ by a merging operation. States $q_1$ and $q_2$ are $\mu$-compatible if

$$\frac{D_{KL}(PA_0, PA_2) - D_{KL}(PA_0, PA_1)}{|PA_1| - |PA_2|} < \mu \tag{4.14}$$

where $|PA|$ denotes the number of states of $PA$ and $D_{KL}(.,.)$ is the Kullback-Leibler divergence [86] defined as $D_{KL}(P_1, P_2) = \sum_{e \in \Omega} P_1[e] \log(P_1[e]/P_2[e])$. That is, $PA_2$ is the new temporary model if the divergence increment relative to the automaton size reduction is less than $\mu$. An efficient computation of the divergence increment resulting from a merging is presented in [125].

We now briefly mention learnability results concerning probabilistic automata. The ALERGIA algorithm has been slightly reformulated under the name RLIPS to *identify in the limit with probability one* the class of probabilistic languages generated by PDFA [27]. This identification framework essentially requires the induction technique to find exactly the structure of the target machine in finite time. In contrast with RLIPS, MDI has not been proved to identify distributions generated by PDFA. However, empirical results in the domain of language modeling show that MDI outperforms ALERGIA [125]. Recently, it has been shown that the class of distributions generated by PDFA is efficiently learnable in the Probably Approximately Correct (PAC) framework [33]. Learning results are often harder to obtain in this framework as it imposes the induction technique to run in a polynomial time of (the inverse of) parameters of a probabilistic bound and of the number of parameters defining the target distribution. The positive PAC learnability result for PDFA requires to define two extra parameters characterizing the sample complexity: (i) a bound on the expected sequence length and (ii) a bound on the distinguishability between states.

The class of distributions generated by PNFA (strictly including distributions generated by PDFA) has been shown to be identifiable in the limit with probability one [37]. However, according to the results of Abe and Warmth [1], there exists no efficient algorithm for learning this class of distributions. Let us recall that PNFA and HMMs model the same class of distributions (see section 2.2), hence these learning results also apply to HMMs. Besides, a recent work [56] has shown that even more general models, called Multiplicity Automata (MA), have interesting learning properties.

MA are relaxed PNFA such that the numerical parameters take their values in $\mathbb{R}$ rather than in $[0, 1]$, furthermore, stochastic or properness constraints need not be satisfied. MA can generate probabilistic languages (or equivalently distributions on $\Sigma^*$) called Rational Stochastic Languages (RSL). This class of languages strictly includes languages generated by PNFA. Furthermore, such languages admit a canonical representation from which follows an efficient learning algorithm called DEES. This technique identifies in the limit RSL but the intermediate languages, obtained before identifying the target, need not define proper distributions on strings (see definition 24). In practice, even if the target language is probabilistic, DEES can return a non-probabilistic language as the learning sample may be insufficient to exactly identify the target. These non-probabilistic languages are called Pseudo-Stochastic Languages (PSRL). The authors of [56] propose a technique for converting such a language into a probabilistic one but it is not guaranteed to be a RSL. Another serious drawback of this approach is that one cannot decide whether two MA define the same PSRL. Experimental results show the superiority of DEES with respect to the ALERGIA and MDI algorithms.

### 4.2.2 Bayesian state merging

In contrast with preceding merging techniques, the Bayesian state merging algorithm due to Stolcke [121] applies to the general class of HMMs rather than to PDFA. This target class is more general than PDFA since the distributions generated by PDFA are properly included in the distributions generated by HMMs. In his thesis [121], Stolcke proposed several alternative induction techniques. We review here the technique applied in his experimental work. The Bayesian state merging method is initialized with a trivial model $H_0$ consisting of linear paths; each one representing a training sequence in the sample $S$. Given a sequence $s \in S$, its representing path in $H_0$ contains $|s|$ states and the emission probability of each state is 1 for the corresponding symbol. Additionally, an initial silent[3] state $q_0$ leads to the first state of each path with probability $1/|S|$ and the last state of each path is connected to a final silent state $q_f$ with probability one. Such an initial model constructed from the sample $S = \{\texttt{abbab}, \texttt{baba}\}$ is displayed in Figure 4.2.

[3] A silent state is a state emitting no symbol.

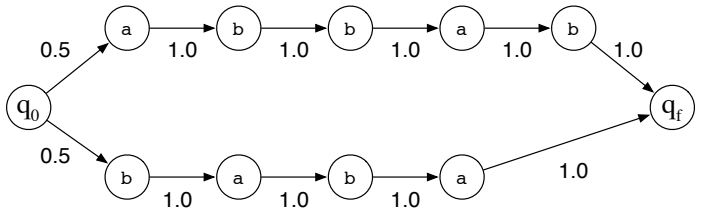



Figure 4.2: The initial model in the Bayesian state merging algorithm applied on the sample $S = \{\texttt{abbac}, \texttt{baba}\}$.

In the sequel, $\rho$ denotes the probabilistic parameters of the model while $H_S$ denotes its structure, *i.e.* its transition graph. The objective of the learning algorithm is to maximize the posterior log-probability of the model structure given the training sample (formally presented later in equation (4.20)). The prior for a model $H$ is defined as a combination of priors for the probabilistic parameters and for the global model structure: $P[H] = P[H_S]P[\rho|H_S]$. For each state $q \in Q$, the outgoing transition probabilities as well as the emission probabilities define multinomial distributions. A natural prior for such distributions is the *Dirichlet distribution*. The Dirichlet prior over the parameters $\boldsymbol{\theta}$ of a multinomial distribution $\mathcal{P}_{\boldsymbol{\theta}}$ is defined as follows

$$P[\boldsymbol{\theta} \mid \boldsymbol{\psi}] = \frac{1}{b(\boldsymbol{\psi})} \prod_{i=1}^{k} \theta_i^{\psi_i - 1} \tag{4.15}$$

where $\boldsymbol{\psi}$ is a $k$-dimensional vector defining the parameters of the Dirichlet distribution and $b(.)$ is the $k$-dimensional beta function (for an accurate implementation of the beta function see [104]). The parameters $\boldsymbol{\psi}$ of the Dirichlet prior defines a prior bias on the multinomial parameters such that $\mathbb{E}[\theta_i \mid \boldsymbol{\psi}] = \psi_i/\psi_{tot}$ where $\psi_{tot} = \sum_{i=1}^{k} \psi_i$. Interestingly, the Dirichlet distribution is a *conjugate* prior for the multinomial distribution. In other words, its functional form[4] is the same as the likelihood function for multinomial distributions. The induction algorithm takes benefit from this property in computing the posterior of a model structure. This computation requires to integrate the product of the multinomial likelihood function and of the Dirichlet prior over all possible values of $\boldsymbol{\theta}$. This integral can be computed in closed form as follows:

$$\int_{\boldsymbol{\theta}} P[\boldsymbol{c} \mid \boldsymbol{\theta}] P[\boldsymbol{\theta} \mid \boldsymbol{\psi}] d\boldsymbol{\theta} = \frac{b(\boldsymbol{c} + \boldsymbol{\psi})}{b(\boldsymbol{\theta})} \tag{4.16}$$

where $\boldsymbol{c}$ is a $k$-dimensional vector such that $c_i$ is the number of times the $i$-th outcome has been observed. While performing a MAP estimation of a

[4]More precisely, the functional form of the numerator.

multinomial parameter $\theta_i$ from counts $\boldsymbol{c}$, the effect of the Dirichlet prior is equivalent to adding $\psi_i - 1$ virtual count(s) to $c_i$, provided $\psi_i \geq 1$:

$$\hat{\theta}_i = \frac{c_i + \psi_i - 1}{\sum_{j=1}^{k} c_j + \psi_j - 1} \tag{4.17}$$

Using $0 < \psi_i < 1$ adds a bias towards the extreme values $\theta_i = 0$ or $\theta_i = 1$ while using $\psi_i = 1$ for all $i = 1, \ldots, k$ defines a uniform prior and the MAP estimate becomes equivalent to the maximum likelihood estimate. In his experiments, Stolcke defines $\psi_{tot} = 1$, leading to a bias towards non-uniform multinomial distributions. All the multinomial distributions defined within a model are considered to be independent. Consequently, the prior of the probabilistic parameters given the model structure is the product of the priors for all multinomials in the model:

$$P[\rho \mid H_S] = \prod_{\boldsymbol{\theta} \in H_S} P[\boldsymbol{\theta} \mid \psi(\boldsymbol{\theta})] \tag{4.18}$$

where $\psi(\boldsymbol{\theta})$ returns the parameters of the Dirichlet priors related to the multinomial parameters $\boldsymbol{\theta}$. The structural prior $P[H_S]$ is defined according to the Minimum Description Length (MDL) principle as

$$P[H_S] = \prod_{q \in Q} (|Q| + 1)^{-n_t(q)} (|\Sigma| + 1)^{-n_e(q)} \tag{4.19}$$

where $n_t(q)$ and $n_e(q)$ are respectively the number of outgoing transitions from state $q$ and the number of distinct symbols emitted by $q$ with a strictly positive probability. This prior adds a bias towards small models as for the MDI algorithm.

The induction algorithm iteratively merges state pairs until reaching a local maximum of the log-posterior of the model structure given the sample:

$$\log(P[H_S \mid S]) = \log(P[S \mid H_S]) + D \log(P[H_S]) \tag{4.20}$$

where $P[S \mid H_S]$ is the likelihood of the sample $S$ given the model structure and $D$ is a hyperparameter, called the *global prior weight*, defining the relative importance of the structure prior versus the likelihood. An intuitive interpretation of this hyperparameter can be given by multiplying[5] the right part of equation (4.20) by $1/D$: $\log(P[H_S]) + \log(P[S \mid H_S]^{1/D})$. The quantity $1/D$ acts as a "data multiplier" and consequently $|S|/D$ is the *effective*

[5]Multiplying the objective function by a positive constant has no effect on the maximization.

*sample size*. This hyperparameter is particularly useful when the induction algorithm is applied online. In such a case, the training sequences are successively presented to the algorithm. The importance of the likelihood should become progressively higher as the model estimation relies on more data. To do so, Stolcke proposes to fix the effective sample size to a given value. Consequently, the parameter $D$ is implicitly decreased as more sequences are available. The probability $P[S \,|\, H_S]$ of the data given the model structure is obtained by integrating the product $P[S \mid H_S, \rho]P[\rho \mid H_S]$ over all possible values of the probabilistic parameters $\rho$:

$$P[S \mid H_S] = \int_{\rho} P[S \mid H_S, \rho]\; P[\rho \mid H_S] d\rho \tag{4.21}$$

Such an integral is very difficult to compute in practice. To circumvent this issue, two assumptions are made: (H1) sequences are generated along their Viterbi path (see section 2.3.2) and (H2) Viterbi paths remain unchanged by a modification of the probabilistic parameters $\rho$. We should note that the second assumption is somewhat crude as the Viterbi path related to a sequence is mostly determined by the probabilistic parameters along that path. The integral (4.21) can be approximated by

$$P[S \mid H_S] \simeq \prod_{\boldsymbol{\theta} \in H_S} \int_{\boldsymbol{\theta}} P[c(\boldsymbol{\theta}) \mid \boldsymbol{\theta}] P[\boldsymbol{\theta} \mid \psi(\boldsymbol{\theta})] d\boldsymbol{\theta} \tag{4.22}$$

where $c(\boldsymbol{\theta})$ denotes the counts relative to the multinomial parameters $\boldsymbol{\theta}$. The inner integrals can be computed in closed form by applying formula (4.16). At the beginning, the counts are equal to one for the parameters present (*i.e.* having a strictly positive probability) in the initial model. Thanks to assumptions (H1) and (H2), these counts are readily updated: when two states $q_1$ and $q_2$ are merged into state $q$, the counts associated to $q$ are obtained by summing up the counts relative to $q_1$ and $q_2$. At each iteration, all pairs of states are considered as candidate for merging. The candidate model leading to the largest improvement of the objective function is kept. The algorithm is iterated until the objective function is no longer improved.

This induction algorithm has several issues. First, at each iteration, $\mathcal{O}(|Q|^2)$ candidate pairs for merging have to be evaluated. If the training sample contains numerous sequences, the early stages of the algorithm may be very slow. To circumvent this issue, Stolcke proposes an online alternative to his technique. The sequences are presented to the algorithm in small batches, typically containing from 1 to 10 sequences. For each sequence in the batch, a linear path is added to the model as depicted in Figure 4.3.

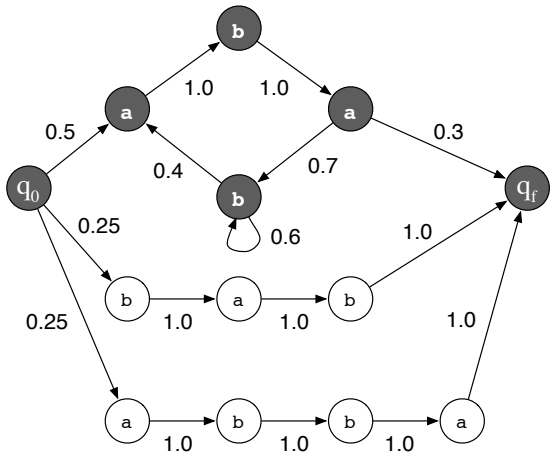



Figure 4.3: The Bayesian state merging due to Stolcke applied online. Here, a batch of two sequences $\{\texttt{bab}, \texttt{abba}\}$ is added to the current model. Filled states denote states of the current model while unfilled states are associated to the new sequences.

Merging operations are carried out until finding a maximum of the objective function and a new batch of sequences is presented to the algorithm. This procedure is iterated until the complete sample is exhausted. The global prior weight is implicitly defined by fixing the effective sample size to a prescribed value, preventing overgeneralization at early stages. Another technique to reduce the computational cost is to only allow merging of states emitting the same symbol at the early stages of the induction. If this constraint is maintained till convergence, the resulting model is a *Partially Observable Markov Model* (POMM, see section 5.2). Another possible issue can happen when the stopping criterion (*i.e.* the objective function is no longer improved) is triggered too early, as sometimes several merging operations are required to improve the posterior of the model structure. To face this problem, the stopping criterion is modified to trigger after a fixed number of steps during which no improvement has been made.

### 4.2.3 Maximum likelihood state splitting

This section presents the maximum likelihood state splitting induction algorithm due to Ostendorf and Singer [99]. This approach mainly focuses on the automatic design of left-to-right models with continuous emission densities, typical in speech recognition. This work extends a previous state splitting technique [123] for allophone modeling. However, it can readily

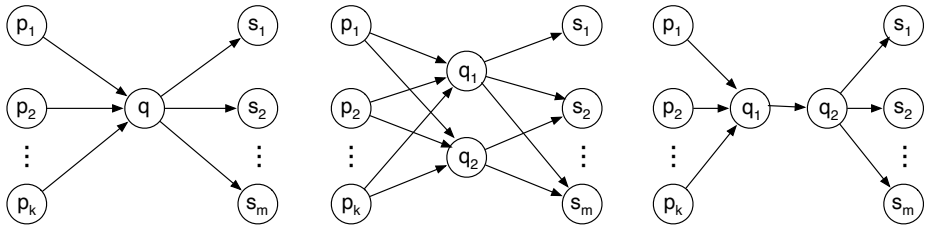



Figure 4.4: State splitting in the maximum likelihood state splitting algorithm due to Ostendorf et al. [99]. Left: state $q$ with its neighbors before splitting. Center: state $q$ and its neighbors after a *contextual split*. Right: state $q$ and its neighbors after a *temporal split*.

be applied to the case of discrete emissions considered in this thesis. An initial model topology is defined and parameters are estimated using the Baum-Welch algorithm (see section 2.3.3). States are then iteratively split in order to maximize the likelihood of the training sequences. The model estimation is solely driven by the likelihood maximization as no priors on the model structure or on parameters are defined. At each iteration, a state $q$ is split into two new states $q_1$ and $q_2$. Two kinds of splitting operations, called here *splitting types*, are considered: (i) contextual split (displayed in the center part of Figure 4.4) and (ii) temporal split (pictured in the right part of Figure 4.4). Note, that in a left-to-right structure the predecessors and the successors of a state $q$ are necessarily disjoint. A contextual split refines the dependencies between the predecessors and successors of state $q$ while a temporal split is used to adapt the process behavior along time in the neighborhood of $q$. In addition, two constraints are imposed for a state $q$ to be split: (i) there must be a sufficient number of sequences flowing through state $q$ to ensure reliable estimations of the parameters relative to the resulting states $q_1$ and $q_2$ and (ii) there must be a path in the model, the length of which accommodates the shortest training sequence. It is important to note that this structural induction technique is not totally general as it cannot induce cyclic model, unless cycles were present in the initial model. In contrast with a previous state splitting technique [123], the splitting type is determined for each candidate state before the actual state to split is selected. To do so, Ostendorf et al. propose efficient tests, not detailed here, to determine the optimal splitting type for a given state. Hence, once the optimal state is selected, one already knows how to update the model structure.

The parameters of the model resulting from the split are estimated by maximum likelihood. More precisely, the expected log-likelihood of the

training sequences is maximized:

$$\begin{aligned}
& \mathbb{E}\left[\log(P[S,\mathcal{H}]) \mid \rho\right] = \sum_{\mathcal{H}} P[\mathcal{H} \mid S,\rho] \log(P[S,\mathcal{H} \mid \rho]) \\
= \;& \log \prod_{q\in Q} \left[ \iota(q)^{\overline{S}(q)} \prod_{\mathtt{a}\in\Sigma} B(q,\mathtt{a})^{\overline{V}(q,\mathtt{a})} \prod_{q'\in Q} A(q,q')^{\overline{N}(q,q')} \right] \\
= \;& \sum_{q\in Q} \overline{S}(q) \log(\iota(q)) + \sum_{\mathtt{a}\in\Sigma} \overline{V}(q,\mathtt{a}) \log(B(q,\mathtt{a})) + \sum_{q'\in Q} \overline{N}(q,q') \log(A(q,q'))
\end{aligned} \tag{4.23}$$

where $\mathcal{H}$ denotes the paths along which the observations $S$ are generated, and $\overline{S}(q), \overline{V}(q,\mathtt{a})$ and $\overline{N}(q,q')$ denote respectively the expectations of the number of times the parameters $\iota(q), B(q,\mathtt{a})$ and $A(q,q')$ are triggered given the training sequences, as defined in section 2.3.3. The form of equation (4.23) allows one to easily determine the contribution of each state in the expected log-likelihood. At each split, only the parameters affected by the topology change, *i.e.* transitions from and to $q_1, q_2$ and the emission distributions relative to these states, are (re)estimated using Baum-Welch. Furthermore, additional constraints impose that the expectations $\overline{S}(.), \overline{V}(.,.)$ and $\overline{N}(.,.)$ relative to unaffected parameters remain unchanged. These constraints are more easily satisfied for left-to-right models since modifying the parameters of a state $q$ does not affect the expectations relative to the predecessors of $q$ (*i.e.* no retro-propagation loop has to be taken into account). Consequently, the log-likelihood gain, used to select the state to split, is obtained by subtracting the contribution relative to $q$ (before being split) from the cumulative contributions relative to $q_1$ and $q_2$. The induction algorithm is iterated until the likelihood gain falls below a user-defined threshold or there are no more allowed states to split. The optimal model size is selected on an independent validation set of sequences. Additional criteria for state splitting, including a $\chi^2$ goodness-of-fit test as well as a Minimum Description Length prior are presented in [120].

## 4.3 Discriminative induction techniques

This section presents an overview of some discriminative learning techniques for HMMs or HMM-related models. Such approaches are typically used to solve sequence classification or sequence labeling problems. In contrast with the techniques presented above, the goal is not to model the generative process of the data but to estimate models with a good discrimination efficiency. Several discriminative criteria have been proposed including the Maximum Mutual Information (MMI) [6], the Minimum Classification Error (MCE) [70, 111, 61], the conditional likelihood [11, 65, 113] and margin-based crite-

ria [34, 127]. From the theoretical point of view, the conditional likelihood criterion is perhaps the most appealing for classification since, according to the Bayes decision theory, the optimal classification follows from maximizing this criterion. Subsection 4.3.1 discusses the maximization of the HMM conditional likelihood. Besides, margin-based techniques and related approaches such as conditional random fields are now considered as the state-of-the-art in sequence labeling tasks. Such approaches are briefly overviewed in subsection 4.3.2.

### 4.3.1 Conditional likelihood based techniques

The classical Baum-Welch algorithm and more generally EM attempt to maximize the probability of the sample $S$ given the model parameters $\rho$: $P[S \mid \rho]$. This kind of estimation is interesting to shed light on how the observations were generated. On the other hand, the estimated models are frequently used in a discriminative setting such as sequence classification. To do so, a model is estimated for each class $y \in \mathcal{Y}$, hence maximizing the joint probability $P[S, Y \mid \rho]$. Given a sequence $s$ to classify, the class having the largest posterior probability is predicted using the Bayes rule: $y^* = \arg\max_y P[s \mid y, \rho]P[y]$. It is well-known that estimating the conditional distribution $P[Y \mid S, \rho]$ through the estimation of the joint distribution $P[S, Y \mid \rho]$ yields often suboptimal discrimination results. In contrast, discriminative techniques have been proposed to directly optimize $P[Y \mid S, \rho]$.

The EM algorithm can be applied to maximize the conditional likelihood [65], however, the Maximization (M) step has no closed form solution. Consequently, this step has to be solved using a gradient ascent technique, making the method computationally intensive. A recent work [113] proposes a more tractable EM-based procedure for maximizing the conditional likelihood. However, this speed-up is obtained by sacrifying the monotonic increase of the conditional likelihood at each iteration. The authors provide reestimation formula for the parameters of HMMs and show that their technique compares favorably with an extended EM algorithm [54] in sequence classification problems.

Alternatively to EM-based techniques, one often resorts to gradient-based algorithms (see [11]). As mentioned in section 2.3.3, these techniques are not as simple as Baum-Welch as the learning rate has to be chosen empirically. This inconvenience can be circumvented by using second-order methods. However, such methods require to compute the inverse of the Hessian matrix (containing the partial second derivatives of the likelihood function at the current solution for all pair of variables), yielding a high computational cost.

### 4.3.2 Margin-based techniques

Several kernel methods have been proposed to model sequential processes. These methods do not explicitly induce a HMM but they are based on features inspired by HMMs and often resort to techniques such as the Viterbi decoding (see section 2.3.2) typically used with HMMs. These techniques mainly focus on the supervised *label sequence learning problem*: predict a sequence of labels associated to an input sequence. Training instances consist of pairs $\{(s^i, y_i), \ldots, (s^n, y_n)\}$ where $s^i$ is sequence of symbols (called the input sequence) and $y_i$ is a sequence of labels (called the label sequence). This setting is different from sequence classification where a single label is predicted from an input sequence. Applications of such techniques include Part-Of-Speech (POS) tagging, Named Entity Recognition (NER) and gene finding, to name a few.

Several techniques to the label sequence learning problem have been proposed including Conditional Random Fields (CRFs) [78], Maximum Entropy Markov Models (MEMMs) [89], Perceptron reranking [34] and Max-Margin Markov ($M^3$) Networks [124]. The basic commonality between these methods is that the label prediction at a given position can depend on past and future features of the input sequence. Hidden Markov SVM (HMSVM) [3] follows this line while adding the maximum margin principle as well as the kernel trick. This work can be instantiated in the Structural SVM framework proposed by Tsochantaridis et al. [127]. One of the key idea of this approach is to define a discriminant function $f(s) = \arg\max_{y \in \mathcal{Y}} \{F(s, y) = \langle \boldsymbol{w}, \Phi(s, y) \rangle\}$ where $\mathcal{Y}$ is an output space consisting of all possible label sequences, $\boldsymbol{w}$ is a weight vector defining the (primal) model parameters and $\Phi(.,.)$ is a joint feature representation of inputs and outputs. As for SVM, this explicit feature mapping can be avoided by using a kernel function over joint input-output features. Two kinds of features are considered: (i) features modeling interaction between the labels and the symbols of input sequences and (ii) features describing local dependencies between the labels along sequences. As for HMMs, the dependencies between labels (corresponding to the dependencies between states in the hidden process (see definition 21)) respect the order 1 Markov property, *i.e.* the prediction of label at time $t$ only depends on the label at time $t-1$. This allows one to compute the $\arg\max_{y \in \mathcal{Y}}$ efficiently using a Viterbi-like decoding algorithm, provided $\boldsymbol{w}$ is known. The notion of margin for an instance $(s^i, y_i)$ is defined as $\gamma_i = F(s^i, y_i) - \max_{y \neq y_i} F(s^i, y)$. The maximization of the margin amounts to finding a weight vector $\boldsymbol{w}$, or equivalently a dual SV expansion of $\boldsymbol{w}$, that maximizes $\min_i \gamma_i$. Such a problem can be efficiently solved using a cutting plane algorithm introducing successively constraints which corre-

spond to the largest margin violations.

Another work [136] deals with the *unsupervised* label sequence learning problem. In contrast with the previous setting, training data are solely made of (unlabeled) sequence of symbols. A discriminative criterion based on the margin maximization is informally defined as follows: strongly different input sequences should be associated to widely separated label sequences. The resulting problem turns out to be a convex optimization problem that can be solved using Semidefinite Programming (SDP). Unfortunately, this problem is very expensive to solve in practice. The authors of [136] propose an approximation technique which substantially reduces the computational cost but it is no longer guaranteed to find the global solution. Even using this approximation, the learning technique can only deal with small datasets (containing about 50 sequences) made of short sequences (containing about 10 symbols).

## 4.4 Summary and discussion

In this chapter, we have presented an overview of techniques for learning Hidden Markov Models (HMMs). These models are more powerful than standard MC as they can define unbounded dependencies while MC rely on a bounded history of past events. This expressiveness increase makes these models harder to train. The estimation of HMMs from data is twofold: (i) the model structure, *i.e.* the number of states and the presence of transitions between these states, has to be defined and (ii) the probabilistic parameters of the model have to be estimated. The structural design is a discrete optimization problem while the parameter estimation is continuous by nature. The parameter estimation alone has been shown to be a hard problem [1]. In other words, the Baum-Welch algorithm for estimating these parameters (see section 2.3.3) is only guaranteed to find a local maximum of the likelihood function. In most cases, the model structure, also referred to as topology, is defined according to some prior knowledge of the application domain. For instance, in speech recognition the *left-to-right* topology prevails while a linear structure (i.e. a profile HMM) is frequently used to model the primary structure of proteins. However, automated techniques for designing the HMM topology are interesting as the structures are sometimes hard to define *a priori* or need to be tuned after some task adaptation. Several approaches towards this objective have been discussed.

A first strategy to learn the model structure relies on the estimation of the probabilistic parameters. An initial model with a prescribed topology is

estimated, and the probabilistic parameters of small magnitude are trimmed, *i.e.* structurally removed from the model. In this context, the simplest approach consists in applying the Baum-Welch algorithm on a fully connected topology and in trimming transitions with small probabilities. Several model sizes are considered and the optimal size is selected on validation data or using a BIC or AIC criterion. This structural learning is far from optimal. For each considered model size, the parameters have to be estimated from scratch which can be highly time consuming. Moreover, we will see in section 5.4 that the Baum-Welch procedure, applied on fully connected graphs, is badly suited to model long-term dependencies.

A more sophisticated technique, due to Brand [16], estimates the parameters in a MAP setting including entropic priors. These priors strongly bias the model towards sparsity. That is, weakly supported parameters are numerically pushed to very small probabilities and can then be trimmed off from the model. The same prior distribution is applied to all the multinomial parameters in the model. This can be inappropriate if the target model has an underlying graph with variably dense regions. Moreover these priors do not take the possible relations between the model parameters into account as they are applied independently. A recent work [84] also uses entropic priors to learn of left-to-right HMMs. This technique attempts to learn the number of states of the model not the (left-to-right) transition graph itself.

State merging techniques start with a complex model perfectly matching the sample distribution and successively merge states to improve the generalization ability of the model. These techniques have been first used in grammatical inference for learning non-probabilistic regular languages. These languages are conveniently represented using Deterministic Finite state Automata (DFA) or Non-deterministic Finite state Automata (NFA). While these two formalisms define the same class of languages in a non-probabilistic setting, their probabilistic counterparts PDFA and PNFA generate distinct classes of distributions. More precisely, distributions generated by PDFA form a proper subclass of the distributions generated by PNFA. Since HMMs are equivalent to PNFA (see section 2.2), PDFA form a proper subclass of HMMs.

Techniques for learning PDFA using state merging include the ALERGIA [26, 27] and MDI [125] algorithms. These two techniques successively merge the states of a probabilistic prefix tree automaton representing the sample distribution. ALERGIA (more precisely a slight adaptation called RLIPS) has been shown to identify in the limit with probability one the class of distributions generated by PDFA. Such a proof has not been provided for MDI, however experimental results in language modeling show the practical superiority of this approach over ALERGIA. Recently, Clark et al. [33] have

shown that a state merging based technique (close to ALERGIA) can learn efficiently distributions generated by PDFA in the PAC framework.

The Bayesian state merging due to Stolcke [121] is perhaps the most well-known HMM structural induction technique. In contrast with previous merging techniques it applies to the general class of HMMs. This technique learns a HMM while maximizing the posterior probability of the model structure. Two kinds of priors are considered: (i) structural priors concerning the transition graph of the model and (ii) parameter priors concerned with the probabilistic parameters of the model. Structural priors are defined according to the Minimum Description Length (MDL) principle while Dirichlet priors are defined over the multinomial parameters of the model. At each iteration, all the candidate pairs are considered and merging is applied to the pair leading to the largest structure posterior increase. The computation of the structure posterior relies on two assumptions: (i) sequences are generated along their maximal probability path (aka the Viterbi path) and (ii) these paths do not change as the model parameters vary. The second assumption is rather crude and is obviously not satisfied in practice. Apart from these modeling assumptions, one of the major drawbacks of this technique is its high computational cost. To face this issue, Stolcke has proposed an on-line version where batches of sequences are successively presented to the model. While reducing the computational cost, this alternative adds more parameters to tune, namely the effective sample size for preventing unsupported merging and the batch size. Finally, this approach has not been shown to clearly outperform alternative techniques such as the Baum-Welch algorithm applied to fully connected graphs. This is verified in our experimental work (see section 5.6).

State splitting methods are initialized with a small initial model and iteratively split states to better fit the training data. Ostendorf et al. [99] have proposed a technique based on state splitting to learn left-to-right HMMs. An initial model topology is defined, the parameters of which are estimated using the Baum-Welch algorithm. States are then iteratively split in order to maximize the likelihood of the training sequences. At each iteration a state is split into two new states. Two types of splitting are considered: (i) a contextual split replaces a state by a pair of parallel states and (ii) a temporal split substitutes a given state by a sequence of two states. The optimal splitting type is determined for all states, via efficient tests, before updating the model structure. This technique is not completely general as it cannot induce cyclic models unless cycles were already included in the initial topology.

A recent research [56] has been proposed to induce Multiplicity Automata (MA) which are more powerful than HMMs. MA can generate probabilistic languages called Rational Stochastic Languages (RSL) which strictly include the distributions generated by HMMs. An efficient algorithm, called DEES, has been proposed to identify RSL in the limit, however, the intermediate languages obtained before identification need not define a proper distribution on strings. These non-probabilistic languages can be converted into probabilistic languages called Pseudo-Stochastic Rational Languages (PSRL). One of the major drawbacks of this approach is that one cannot decide whether two MA define the same PSRL.

Discriminative learning approaches build models optimizing the conditional likelihood, *i.e.* the class posterior, or a discriminant function. That is, models are trained to solve a specific, discriminative, task such as sequence classification or labeling. Such model does not embed the "complexity" necessary for generating the training sequences but the "complexity" required to discriminate these sequences. The conditional likelihood associated to HMMs can be optimized using adapted EM algorithms [65, 54] or by applying gradient-based techniques [11]. On the one hand, conditional EM algorithms are more complex than the standard Baum-Welch procedure as the Maximization step has no closed form solution. On the other hand, gradient-based techniques require to tune the learning rate and to take the properness constraint into account explicitly (EM automatically satisfy these constraints). Besides, margin-based techniques [34, 127] and related approaches such as conditional random fields [78] are now considered as the state-of-the-art in sequence labeling tasks. Margin-based techniques extend Support Vector Machines (SVM) by considering structured outputs which are here sequences of labels. Strictly speaking, these techniques do not apply to HMMs but rely on features inspired from HMMs such as the Markovian dependencies between the labels (corresponding to hidden states in HMMs) in the output sequences.

In chapter 5, we propose a novel approach, called POMMStruct, to the structural induction of HMMs. POMMStruct learns the structure and the parameters of Partially Observable Markov Models (POMMs), which are equivalent to HMMs, to model the First Passage Times (FPT) of the target process. It learns a model by iteratively adding new states and estimating the additional parameters to best fit the observed FPT. This approach circumvents several issues related to the techniques discussed above. First POMMStruct deals with the general class of HMMs unlike ALERGIA or MDI which apply to the restricted class of PDFA. In contrast with induction approaches relying on parameter estimation, POMMStruct does not

apply to a fixed size model. In particular, states can be structurally added and the augmented model needs not be reestimated from scratch. Moreover it is not restricted to left-to-right topologies like the approach of Ostendorf et al. Unlike Baum-Welch applied to fully connected topologies, POMMSTRUCT is well-suited to model long-term dependencies. POMMSTRUCT also proposes a novel trimming strategy which is based on the expected number of times transitions are used rather than on the transition probabilities. This criterion takes the global dynamics of the model into account whereas trimming according to transition probabilities is only based on the local dynamics of the model. POMMSTRUCT is shown to outperform the Bayesian merging technique of Stolcke in section 5.6.

# Chapter 5

# Learning HMMs from First Passage Times

*This chapter presents a novel approach to the structural induction of Hidden Markov Models (HMMs). The induction algorithm, called* POMMStruct*, aims at estimating a model that accurately fits the First Passage Times (FPT) between symbols in sequences.* POMMStruct *applies to* Partially Observable Markov Models *(POMMs) which are graphical models equivalent to HMMs. More precisely, POMMs are HMMs for which any state emits a single symbol but the same symbol can be emitted by several states. The state set of a POMM can be partitioned into* blocks *by gathering the states emitting the same letter. These models are called* partially observable *since the observation of a sequence emitted by a POMM identifies the blocks reached by the process but not precisely which states inside the blocks. We show that POMMs and HMMs are equivalent in the sense that they generate the same probabilistic languages (see definition 24). A constructive proof of this equivalence is provided by showing how a given HMM can be transformed into an equivalent POMM. The relations between standard MC and POMMs are also investigated. When all the states emit a distinct symbol, the process is fully observable (i.e. an observation corresponds to a unique state) and is equivalent to an order 1 MC. Otherwise, we show that a POMM is equivalent to a lumped MC.*

*Next, the distributions of FPT in POMMs (equivalently in HMMs) are studied. We show that these distributions are of phase-type. The FPT dynamics is a fundamental characteristic of random walks in POMMs. Indeed, POMMs are more powerful models than standard MC, and their FPT dynamics is generally poorly approximated by local models. We motivate the use of the FPT in* POMMStruct *by showing that these features are informative about the topology to be learned. Furthermore, we show that a HMM*

*with an appropriate structure can model long-term dependencies and that these dependencies are reflected in the FPT dynamics. We also argue why an accurate modeling of long-term dependencies cannot be achieved through the classical approach of Baum-Welch estimation of a fully connected model.*

*Starting from a standard MC,* POMMStruct *iteratively adds states to the model. The probabilistic parameters are estimated using a novel method based on the EM algorithm, called* POMMPHit*. It computes the POMM parameters that maximize the likelihood of the FPT observed in the training sequences. This algorithm differs from the standard Baum-Welch procedure (see section 2.3.3) since the likelihood function to be maximized is concerned with times between events (i.e. emission of symbols) rather than with the complete generative process. A simpler version of this algorithm is also used to model FPT in the context of sequence classification (see section 7.3). An interesting byproduct of the expectation step of* POMMPHit *is the expected transition passage times. It provides the average number of times a given transition is triggered when observing the FPT in the sample. This offers a very natural criterion to trim off less frequently used transitions from the model.* POMMStruct *is iterated until convergence of the FPT likelihood function and the optimal model size is selected on an independent validation set.*

*This chapter is organized as follows. Section 5.1 presents the FPT features extracted from sequences, on which* POMMStruct *relies. Section 5.2 introduces* Partially Observable Markov Models *(POMMs). The equivalence with HMMs and the relations with lumped MC are respectively detailed in subsections 5.2.1 and 5.2.2. The distributions of FPT in POMMs are studied in section 5.3. Section 5.4 is concerned with the modeling of probabilistic long-term dependencies. The poor approximation of such dependencies by local models is illustrated in subsection 5.4.1. Subsections 5.4.2 and 5.4.3 respectively highlight the importance of the HMM topology to fit long-term dependencies and the impact of such dependencies on the FPT dynamics. Section 5.5 presents the structural induction algorithm* POMMStruct *as well as the parameter estimation procedure* POMMPHit*. Finally, experimental results obtained with* POMMStruct *and other competing methods on several sequence modeling tasks are presented in section 5.6.*

## 5.1 First Passage Times in sequences

**Definition 35** *Given a sequence $s$ defined on an alphabet $\Sigma$ and two symbols $\mathtt{a}, \mathtt{b} \in \Sigma$. For each occurrence of $\mathtt{a}$ in $s$, the first passage time to $\mathtt{b}$ is the finite*

*number of steps taken for observing the next occurrence of* $\mathtt{b}$. $\mathrm{FPT}_s(\mathtt{a},\mathtt{b})$ *denotes the first passage times to* $\mathtt{b}$ *for all occurrences of* $\mathtt{a}$ *in* $s$. *It is represented by a set of pairs* $\{(z_1,c_1),\ldots,(z_l,c_l)\}$ *where* $z_i$ *denotes a passage time and* $c_i$ *is the number of times* $z_i$ *is observed in* $s$.

For instance, let us consider the sequence $s = aababba$ defined over the alphabet $\Sigma = \{\mathtt{a},\mathtt{b}\}$. The FPT from $\mathtt{a}$ to $\mathtt{b}$ in $s$ are $\mathrm{FPT}_s(\mathtt{a},\mathtt{b}) = \{(2,1),(1,2)\}$. The potential number of FPT pairs is bounded by $|\Sigma|^2$. In POMMStruct (see section 5.5), the FPT are extracted from a set $S$ of sequences rather than from a single sequence. The *empirical FPT distribution* relative to a pair $(\mathtt{a},\mathtt{b})$ is obtained by computing the relative frequency of each distinct passage time from $\mathtt{a}$ to $\mathtt{b}$. In contrast with $N$-gram features (see section 1.5), the FPT do not only focus on the local dynamics in sequences as there is no *a priori* fixed maximum time (*i.e.* number of steps) between two events. For this reason, such features are well-suited to model long-term dependencies (see section 5.4.3). In section 5.3, we motivate the use of the FPT in the induction algorithm by showing that they are informative about the model topology to be learned.

## 5.2 Partially Observable Markov Models

This section introduces Partially Observable Markov Models (POMMs). These models are used in the structural induction algorithm presented in section 5.5. The equivalence between POMMs and HMMs is detailed in section 5.2.1. The links between POMMs and classical MC are detailed in subsection 5.2.2.

**Definition 36 (POMM)** *A* Partially Observable Markov Model (POMM) *is a HMM* $H = \langle \Sigma, Q, A, B, \iota \rangle$ *with emission probabilities satisfying:* $\forall q \in Q, \exists a \in \Sigma$ such that $B(q,a) = 1$.

In other words, any state in a POMM emits a single symbol but the same symbol can be emitted by distinct states. A POMM is displayed in the right part of Figure 5.1. The *observable partition* $\kappa = \{\kappa_{\mathtt{a}}, \kappa_{\mathtt{b}}, \ldots, \kappa_{\mathtt{z}}\}$ of the state set $Q$ gathers the states emitting the same symbol into blocks.

**Definition 37 (Observable partition)** *Given a POMM* $H = \langle \Sigma, Q, A, B, \iota \rangle$, *the* observable partition $\kappa$ *is made of the blocks* $\kappa_{\mathtt{a}} = \{q \in Q \mid B(q,\mathtt{a}) = 1\}$ *for all* $\mathtt{a} \in \Sigma$.

As for HMMs, the observation of a sequence drawn from a POMM does not allow one to identify the states from which each symbol was emitted. However, we know from which block each symbol has been emitted. For this

reason, these models are called *partially observable*. Whenever each block contains only a single state, the POMM is fully observable and is equivalent to an order 1 MC. It should be pointed out that a POMM still defines a proper HMM. Therefore, the definitions of *underlying MC* (see definition 21), *emission process* (see definition 22) and *HMM path* (see definition 23) as well as the solutions to the three main problems (see section 2.3) also apply to POMMs.

### 5.2.1 Equivalence between POMMs and HMMs

We show here that the class of POMMs is equivalent to the class of HMMs, as any distribution generated by a HMM with $|Q|$ states over an alphabet $\Sigma$ can be represented by a POMM with $\mathcal{O}(|Q|.|\Sigma|)$ states.

**Theorem 8 (Equivalence between HMMs and POMMs)**
*Let $H = \langle \Sigma, Q, A, B, \iota \rangle$ be a HMM, there exists an equivalent POMM $H' = \langle \Sigma, Q', A', B', \iota' \rangle$.*

**Proof.**
*Let $H'$ be defined as follows.*

- $Q' = Q \times \Sigma$,
- $B'((q, \mathtt{a}), x) = 1$ *if* $x = \mathtt{a}$, *and 0 otherwise,*
- $A'((q, \mathtt{a}), (q', \mathtt{b})) = A(q, q')B(q', \mathtt{b})$,
- $\iota'((q, \mathtt{a})) = \iota(q)B(q, \mathtt{a})$.

*It is easily shown that $H'$ satisfies the stochasticity constraints. Let $s = s_0 \dots s_{l-1}$ be a sequence defined on $\Sigma$ and $\nu = \nu_0 \dots \nu_{l-1}$ be a path in $H$. We shall show that $P_{H'}(s, (\nu_0, s_0) \dots (\nu_{l-1}, s_{l-1})) = P_H(s, \nu)$:*

$$\begin{aligned} P_{H'}(s, \nu) &= \iota'(\nu_0)B'((\nu_0, s_0), s_0) \textstyle\prod_{i=1}^{l-1} A'((\nu_{i-1}, \nu_{i-1}), (\nu_i, s_i))B'((\nu_i, s_i), s_i) \\ &= \iota'(\nu_0) \textstyle\prod_{i=1}^{l-1} A'((\nu_{i-1}, \nu_{i-1}), (\nu_i, s_i)) \\ &= \iota(\nu_0)B(\nu_0, s_0) \textstyle\prod_{i=1}^{l-1} A(\nu_{i-1}, \nu_i)B(\nu_i, s_i) = P_H(s, \nu) \end{aligned}$$

*Summing up over all possible paths of length $l$ in $H'$ provides $P_{H'}(s) = P_H(s)$. Hence, $H$ and $H'$ generate the same distribution.*

■

This transformation is illustrated in Figure 5.1. The emission probabilities are omitted as they directly derive from the state labels. There are several

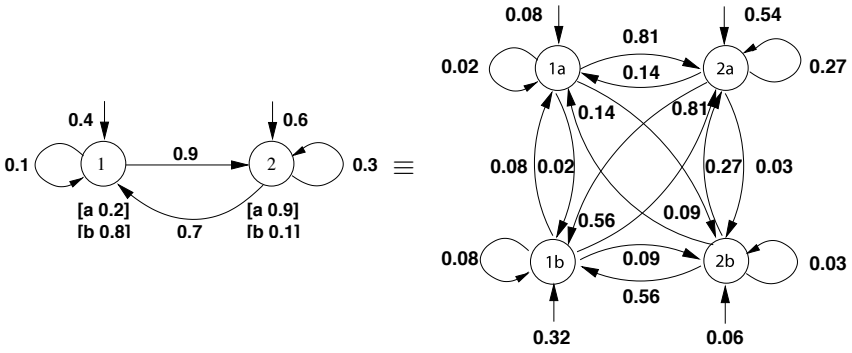



Figure 5.1: Transformation of a HMM into an equivalent POMM.

ways to build a POMM equivalent to a given HMM. We propose a slightly different construction in [21]. The theoretical question of finding a minimal POMM is left for future research. An immediate corollary of this theorem is the equivalence between PNFA and POMMs (see section 2.2). It should be stressed that all transition probabilities of the form $A'((q,\_),(q',b))$ are necessarily equal as the value of $A'((q,a),(q',b))$ does not depend on $a$ in a POMM constructed in this way. A state $(q,a)$ in this model represents the state $q$ after having emitted the symbol $a$ during a random walk in the original HMM.

### 5.2.2 Links between POMMS and Markov chains

In this section, the links between POMMs and MC are highlighted. The material exposed here is not a prerequisite for the induction algorithm presented in section 5.5 but motivates its design. Therefore, the reader mostly interested in structural induction can safely skip this section. In the beginning of section 5.2, we mentioned that when each block of a POMM contains a single state, the process is equivalent to an order 1 MC. In this section, we show that, in general, a POMM is equivalent to a *lumped process* in a MC.

**Definition 38 (Lumped process)** *Given a MC $T = \langle Q, A, \iota\rangle$, let $q_t$ be the state reached at time $t \in \mathbb{N}$, i.e. $X_t = q_t$. Let $\kappa = \{\kappa_1, \kappa_2, \ldots, \kappa_r\}$ denote a partition of the set of states $Q$. The function $K_\kappa : Q \to 2^Q$ maps the state $q$ to the block of $\kappa$ containing $q$. The* lumped process *$T/\!/\kappa$ outcomes $K_\kappa(q_t)$ at time $t$.*

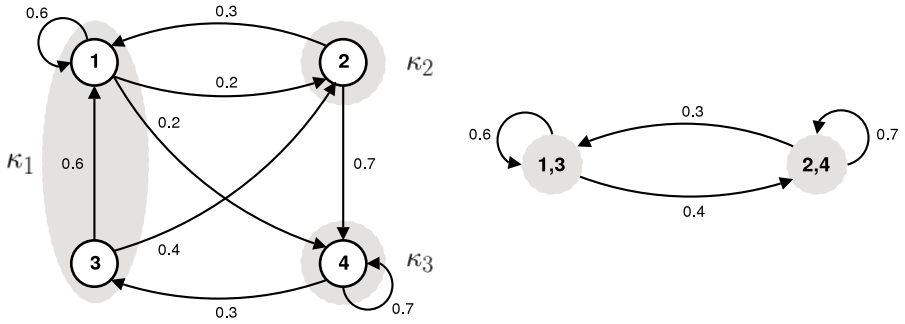



Figure 5.2: Left: A regular Markov chain $T_1$ and the partition $\kappa = \{\{1,3\},\{2\},\{4\}\}$. Right: $T_1$ lumped with respect to the partition $\kappa' = \{\{1,2\},\{3,4\}\}$.

Consider for example the regular MC $T_1$ illustrated[1] in the left part of Figure 5.2. A partition $\kappa$ is defined on its states set, with $\kappa_1 = \{1,3\}, \kappa_2 = \{2\}$ and $\kappa_3 = \{4\}$. The random walk 312443 in $T_1$ corresponds to the following observations in the lumped process $T_1/\!/\kappa$: $\kappa_1\kappa_1\kappa_2\kappa_3\kappa_3\kappa_1$. While the states are fully observable during a random walk in a MC, a lumped process is associated with random walks where only blocks are observed. In this sense, the lumped process makes the MC only partially observable as it is the case for a POMM. The following proposition makes the connection between POMMs and lumped processes.

**Proposition 6** *The emission process $\{O_t \in \Sigma\}_t$ associated to a POMM $H = \langle \Sigma, Q, A, B, \iota \rangle$ with an observable partition $\kappa$ is equivalent to the lumped process $T/\!/\kappa$ where $T = \langle Q, A, \iota \rangle$ is the underlying MC of $H$.*

**Proof.**

*This result is obtained by applying the definition of a lumped process (see definition 38) to the underlying MC of $H$ (see definition 21) with respect to the observable partition $\kappa$ and by noting that there is a bijection between $\Sigma$ and $\kappa$.* ■

It is important to note that the Markov property is not necessarily satisfied for a lumped process and consequently neither for a POMM or a HMM. For example, the lumped MC in the left part of Figure 5.2 satisfies $P[L_t = \kappa_2 \,|\, L_{t-1} = \kappa_1, L_{t-2} = \kappa_2] = 0.2$ and $P[L_t = \kappa_2 \,|\, L_{t-1} = \kappa_1, L_{t-2} = \kappa_3] = 0.4$ where $L_t$ denotes the block reached by the lumped process at time $t$. This clearly violates the *first-order* Markov property. In general, the Markov

[1] For the sake of clarity, the initial probability of each state is not depicted.

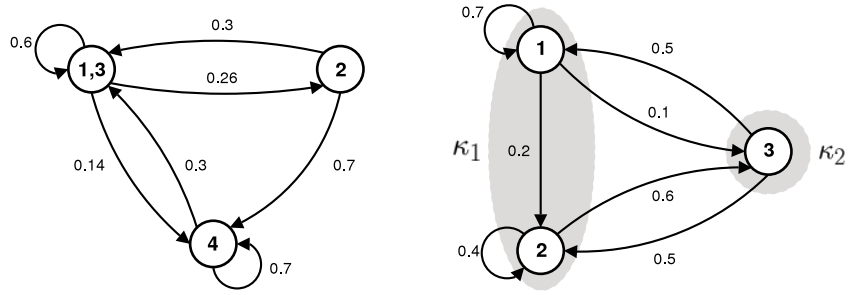



Figure 5.3: Left: The quotient of $T_1$ with respect to the partition $\kappa = \{\{1,3\},\{2\},\{4\}\}$. Right: A non Markovian lumped process.

property is not satisfied when, for a fixed length history, it is impossible to decide unequivocally which state the process has reached in a given block while the next step probability differs for several states in this block. This can be the case no matter the length of the history considered. This is illustrated by the MC depicted in the right part of Figure 5.3 and the partition $\kappa = \{\{1,2\},\{3\}\}$. Even if the complete history of the lumped process is given, there is no way to know the state reached in $\kappa_1$. Thus, the probability $P[L_t = \kappa_2 \,|\, L_{t-1} = \kappa_1, L_{t-2}, \ldots, L_0]$ cannot be unequivocally determined and the lumped process is not Markovian for any order. However, it is interesting to know when a lumped process (or equivalently a POMM) preserves the Markovian property. This notion is called *lumpability*. As mentioned earlier, a sufficient condition for lumpability is that each block of the partition $\kappa$ contains a single state. We state hereafter necessary and sufficient conditions for lumpability.

**Theorem 9 (Necessary and sufficient conditions for lumpability)** *A MC is lumpable with respect to a partition $\kappa$ if and only if for every pair of blocks $\kappa_i$ and $\kappa_j$ the probability $A_{ij}/\!/\kappa$ to reach some state of $\kappa_j$ is equal from every state in $\kappa_i$:*

$$\forall \kappa_i, \kappa_j \in \kappa, \forall q, q' \in \kappa_i,\ A_{ij}/\!/\kappa \triangleq \sum_{q'' \in \kappa_j} A_{qq''} = \sum_{q'' \in \kappa_j} A_{q'q''}$$

*The proof of this result is given in [73].*

The values $A_{ij}/\!/\kappa$ form the transition matrix of the lumped chain. For example, the MC $T_1$ given in the left part of Figure 5.2 is not lumpable with respect to the partition $\kappa = \{\{1,3\},\{2\},\{4\}\}$ while it is lumpable with respect to the partition $\kappa' = \{\{1,3\},\{2,4\}\}$. The lumped chain $T_1/\!/\kappa'$ is illustrated in the right part of Figure 5.2. A related, but possibly different,

process is obtained when the states of the original MC are *merged* to form a *quotient Markov chain*.

**Definition 39 (Quotient MC)** *Given a MC $T = \langle Q, A, \iota \rangle$ and a partition $\kappa = \{\kappa_1, \kappa_2, \ldots, \kappa_r\}$ on $Q$, the quotient $T/\kappa$ is a $r$-states MC with transition matrix $A/\kappa$ and initial vector $\boldsymbol{\iota}/\kappa$ defined as follows:*

$$A_{ij}/\kappa = \sum_{q \in \kappa_i} \sum_{q' \in \kappa_j} \sigma_q^{\kappa_i} A_{qq'}, \quad \iota_i/\kappa = \sum_{q \in \kappa_i} \iota(q)$$

*where $\sigma_q^{\kappa_i}$ is the expected proportion of time the process associated to $T$ reaches state $q$ relatively to the states in $\kappa_i$.*

The left part of Figure 5.3 presents the quotient model of $T_1$ (shown in Figure 5.2) with respect to $\kappa = \{\{1,3\},\{2\},\{4\}\}$. Since $T_1$ is a regular MC (see definition 11), $\boldsymbol{\sigma}^{\kappa_i}$ is computed from the stationary distribution $\boldsymbol{\pi}$ of $T_1$ as follows: $\sigma_q^{\kappa_i} = \pi_q / \sum_{q \in \kappa_i} \pi_q$ for all $q \in \kappa_i$. The stationary distribution of $T_1$ is $\boldsymbol{\pi} = [0.29 \;\; 0.11 \;\; 0.14 \;\; 0.46]^T$. Note that the quotient $T/\kappa$ has always the Markov property while, as mentioned before, this is not necessarily the case for the lumped process $T/\!\!/\kappa$. The following proposition specifies under which condition the distributions generated by $T/\kappa$ and $T/\!\!/\kappa$ are identical.

**Proposition 7** *If a MC $T$ is lumpable with respect to a partition $\kappa$ then $T/\kappa$ and $T/\!\!/\kappa$ define the same process.*

**Proof.**
*When $T$ is lumpable with respect to $\kappa$, the transition probabilities between any pair of blocks $\kappa_i$, $\kappa_j$ are the same in both models:*

$$A_{i,j}/\kappa = \sum_{q \in \kappa_i} \sigma_q^{\kappa_i} \sum_{q' \in \kappa_j} A_{qq'} = A_{ij}/\!\!/\kappa \sum_{q \in \kappa_i} \sigma_q^{\kappa_i} = A_{ij}/\!\!/\kappa.$$

■

To sum up, we have shown here that POMMs are equivalent to lumped MC, *i.e.* MC with a partitioned state set and such that only the blocks of the partition are observed. Such a process has, in general, not the Markovian property as the states cannot be unequivocally identified given an arbitrary long process history. However, in certain cases, a lumped process is equivalent to a MC. Such a process is said to be lumpable with respect to the considered partition. The quotient of a MC with respect to a partition is obtained by merging the states belonging to the same block. In contrast with lumped processes, the resulting process has always the Markovian property. Both processes are equivalent when the original MC is lumpable with respect to the considered partition.

## 5.3 First Passage Times in POMMs

In this section, the distributions of the FPT in POMMs are studied. We show that the FPT distributions between blocks are of phase-type (abbreviated by PH, see section 1.4) by constructing their representing absorbing MC. POMMStruct aims at fitting these PH distributions from the FPT observed in a training sample. We motivate the use of these distributions by showing that they are informative about the model structure to be learned.

First, let us formally define the FPT for a pair of symbols $(\mathtt{a}, \mathtt{b})$ in a POMM.

**Definition 40 (First Passage Times in POMMs)** *Given a POMM* $H = \langle \Sigma, Q, A, B, \iota \rangle$*, the* first passage time *(FPT) is a function* $\mathrm{fpt} : \Sigma \times \Sigma \to \mathbb{N}^0$ *such that* $\mathrm{fpt}(\mathtt{a}, \mathtt{b})$ *is the number of steps for reaching block* $\kappa_{\mathtt{b}}$ *for the first time, starting initially from the block* $\kappa_{\mathtt{a}}$*:*

$$\mathrm{fpt}(\mathtt{a}, \mathtt{b}) = \inf_t \{ t \in \mathbb{N}^0 \mid X_t \in \kappa_{\mathtt{b}} \text{ and } X_0 \in \kappa_{\mathtt{a}} \} \tag{5.1}$$

Let us now compute the probability $P[\mathrm{fpt}(\mathtt{a}, \mathtt{b}) = k]$ with $k \in \mathbb{N}^0$. To do so, one can use absorbing MC techniques presented in section 1.3. In a few words, a reduced absorbing MC (see definition 14) modeling the FPT from $\kappa_{\mathtt{a}}$ to $\kappa_{\mathtt{b}}$ will be set up. The FPT distribution from $\kappa_{\mathtt{a}}$ to $\kappa_{\mathtt{b}}$ in the POMM is equivalent to the PH distribution associated to the time to absorption in this absorbing MC. We assume here that each state $q_0 \in \kappa_{\mathtt{a}}$ leads stochastically (see definition 6) to a state in $\kappa_{\mathtt{b}}$. Otherwise, such a state can be ignored in the following computations.

First, let us compute the starting probability of each state $q_0 \in \kappa_{\mathtt{a}}$ while observing a FPT originated in $\kappa_{\mathtt{a}}$. This probability, denoted by $P_{\mathrm{FPT}}[X_0 = q_0 \mid X_0 \in \kappa_{\mathtt{a}}]$, is obtained by computing the expected proportion of time the process reaches state $q_0$ relatively to the states in $\kappa_a$. According to the nature of the underlying MC $T = \langle Q, A, \boldsymbol{\iota} \rangle$ of $H$, this expectation is computed using distinct techniques from the standard MC theory. If all states in $\kappa_{\mathtt{a}}$ belong to the same final class $\mathcal{C}$ (for instance, see the left part of Figure 5.4), we can restrict our attention to the irreducible subchain $T_{\mathcal{C}}$ induced by $\mathcal{C}$. The left eigenvector $\boldsymbol{\pi}$ of the transition matrix of $T_{\mathcal{C}}$ associated to the eigenvalue 1 contains, for each state $q \in \mathcal{C}$, the expected proportion of the time the process reaches $q$ (see section 1.3). The desired probability is provided by

$$P_{\mathrm{FPT}}[X_0 = q_0 \mid X_0 \in \kappa_{\mathtt{a}}] = \frac{\pi_{q_0}}{\sum_{q \in \kappa_{\mathtt{a}}} \pi_q} \tag{5.2}$$

If the states in $\kappa_a$ are spread over distinct final classes $\mathcal{C}_1, \ldots, \mathcal{C}_r$ (for instance, see the center part of Figure 5.4), one has first to compute the

probability to reach these final classes. The probability $P_{trap}[\mathcal{C}_i]$ to reach the final class $\mathcal{C}_i$ is computed as in equation (1.30) and normalized to get a proper distribution over $\{\mathcal{C}_1, \ldots, \mathcal{C}_r\}$. If $q_0 \in \mathcal{C}_i$ and $\boldsymbol{\pi}$ is computed as above with respect to the subchain induced by $\mathcal{C}_i$, the desired probability is provided by

$$P_{\mathrm{FPT}}[X_0 = q_0 \mid X_0 \in \kappa_{\mathtt{a}}] = P_{trap}[\mathcal{C}_i] \frac{\pi_{q_0}}{\sum_{q \in \kappa_{\mathtt{a}} \cap \mathcal{C}_i} \pi_q} \tag{5.3}$$

If $\kappa_{\mathtt{a}}$ contains both transient and recurrent states (*i.e.* states belonging to a final class), the desired probability is null for transient states and is computed as above for recurrent states. If $\kappa_{\mathtt{a}}$ contains only transient states (for instance, see the right part of Figure 5.4), the probability of starting in state $q_0$ for the FPT originated in $\kappa_{\mathtt{a}}$ is computed from the mean passage times in transient states (see definition 15) :

$$P_{\mathrm{FPT}}[X_0 = q_0 \mid X_0 \in \kappa_{\mathtt{a}}] = \frac{[\boldsymbol{u}^T (I - F)^{-1}]_{q_0}}{\sum_{q \in \kappa_{\mathtt{a}}} [\boldsymbol{u}^T (I - F)^{-1}]_q} \tag{5.4}$$

where $F$ is the transition submatrix between transient states and $\boldsymbol{u}$ is the initial vector for transient states.

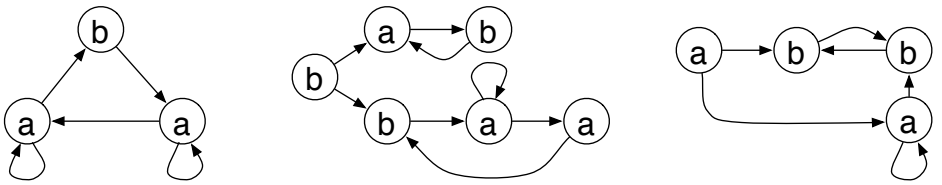



Figure 5.4: Left: All the states in $\kappa_{\mathtt{a}}$ belong to the same final class. Center: the states in $\kappa_{\mathtt{a}}$ are spread over two final classes. Right: the states in $\kappa_{\mathtt{a}}$ are transient.

Let us now construct an absorbing MC modeling the FPT from $\kappa_{\mathtt{a}}$ to $\kappa_{\mathtt{b}}$ in the POMM $H$. An initial distribution $\boldsymbol{\sigma}$ is defined as follows:

$$\sigma_{q_0} = \begin{cases} P_{\mathrm{FPT}}[X_0 = q_0 \mid X_0 \in \kappa_{\mathtt{a}}] & \text{if } q_0 \in \kappa_{\mathtt{a}}, \\ 0 & \text{otherwise} \end{cases} \tag{5.5}$$

All the states in $\kappa_{\mathtt{b}}$ are merged into a single absorbing state $q_{\kappa_{\mathtt{b}}}$. It is assumed here that $\mathtt{a} \neq \mathtt{b}$. Otherwise, a similar construction can be made but the states in $\kappa_{\mathtt{a}}$ have to be duplicated such that the original states are used as starting states and the duplicated ones are merged. The FPT from $\kappa_{\mathtt{a}}$ to $\kappa_{\mathtt{b}}$ can now be studied as the time to absorption in $q_{\kappa_{\mathtt{b}}}$ while starting according

to $\boldsymbol{\sigma}$. It is important to note that the obtained model might contain several final classes. It is therefore not sufficient to compute the (unconditional) time to absorption as the process might be trapped into another final class than $\{q_{\kappa_\mathtt{b}}\}$. The process behavior before specifically getting absorbed into $q_{\kappa_\mathtt{b}}$ can be analyzed using absorbing MC techniques presented in section 1.3. Applying formula (1.34) to the present model results in an absorbing MC $T' = \langle Q_T \cup q_{\kappa_\mathtt{b}}, A', \boldsymbol{\iota}' \rangle$ with the same transient set and a unique final class $\{q_{\kappa_\mathtt{b}}\}$. The obtained initial vector and transition matrix are block partitioned as follows:

$$\boldsymbol{\iota}' = (\boldsymbol{\sigma}^{\kappa_\mathtt{a}}, 0)^T, \quad A' = \begin{pmatrix} F^{\kappa_\mathtt{b}} & \boldsymbol{e}^{\kappa_\mathtt{b}} \\ 0 & 1 \end{pmatrix} \tag{5.6}$$

where $\boldsymbol{\sigma}^{(\kappa_\mathtt{a})}$ is a $|Q_T|$-dimensional vector representing the initial distribution for transient states, $F^{(\kappa_\mathtt{b})}$ is the $|Q_T| \times |Q_T|$ transient submatrix between transient states and $\boldsymbol{e}^{(\kappa_\mathtt{b})}$ is a $|Q_T|$-dimensional absorption vector. The probability that the FPT from $\kappa_\mathtt{a}$ to $\kappa_\mathtt{b}$ occurs after $k$ steps in $H$ is equivalent to the probability to get absorbed in $k$ steps in $T'$:

$$P[\mathrm{fpt}(\mathtt{a}, \mathtt{b}) = k] = (\boldsymbol{\sigma}^{\kappa_\mathtt{a}})^T (F^{\kappa_\mathtt{b}})^{k-1}\, \boldsymbol{e}^{\kappa_\mathtt{b}} \tag{5.7}$$

The previous computations prove the following theorem.

**Theorem 10 (FPT in POMMs are of phase type)** *Given a POMM $H = \langle \Sigma, Q, A, B, \iota \rangle$, the first passage times from $\mathtt{a} \in \Sigma$ to $\mathtt{b} \in \Sigma$ in $H$ are drawn from a phase type distribution represented by the couple $(\boldsymbol{\sigma}^{\kappa_\mathtt{a}}, F^{\kappa_\mathtt{b}})$.*

An immediate corollary of theorems 8 and 10 is that the FPT in HMMs are drawn from PH distributions.

An irreducible POMM $H$ is depicted in the left part of Figure 5.5. The expected proportion of the time the process reaches each state is provided by its stationary distribution : $\boldsymbol{\pi} = (\mathtt{1d} : 0.24, \mathtt{1a} : 0.14, \mathtt{1c} : 0.16, \mathtt{1b} : 0.13, \mathtt{2c} : 0.12, \mathtt{2b} : 0.10, \mathtt{2a} : 0.11)^T$. An absorbing MC $T'$ representing the FPT from $\kappa_\mathtt{a}$ to $\kappa_\mathtt{b}$ is obtained by merging the states `1b` and `2b` into a single absorbing state $q_{\kappa_\mathtt{b}}$. The initial distribution for transient states is set up from $\boldsymbol{\pi}$ restricted to $\kappa_\mathtt{a}$. : $\boldsymbol{\sigma}^{\kappa_\mathtt{a}} = (\mathtt{1d} : 0, \mathtt{1a} : 0.56, \mathtt{1c} : 0, \mathtt{2c} : 0, \mathtt{2a} : 0.44)^T$. This absorbing MC is shown in the center of Figure 5.5. The resulting PH distribution is displayed in the right part of Figure 5.5. This distribution has several modes (*i.e.* maxima), the most noticeable being at times 2 and 4. These modes reveal the presence of paths of length[2] 2 and 4 from $\kappa_\mathtt{a}$ to $\kappa_\mathtt{b}$ having a large probability. For instance, the paths `1a,1c,1b` and

[2]The length of a path is defined here in terms of number of steps.

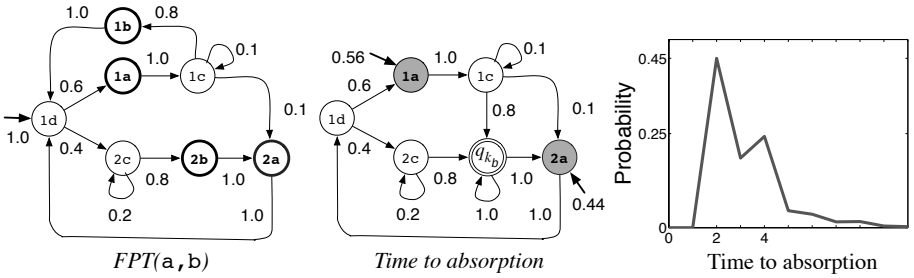



Figure 5.5: Left: an irreducible POMM $H = \langle \Sigma, Q, A, B, \iota \rangle$. Center: an absorbing MC $T'$ representing the FPT from block $\kappa_{\mathtt{a}}$ to block $\kappa_{\mathtt{b}}$ in $H$. The gray filled states are the initial states of the MC and their initial probabilities are shown as well. Right: The PH distribution of the absorption times in $T'$, or equivalently, of the FPT between $\kappa_{\mathtt{a}}$ and $\kappa_{\mathtt{b}}$ in $H$.

`2a,1d,1a,1c,1b` have a respective probability equal to 0.45 and 0.21 (other paths of length 4 yield a total probability equal to 0.25 for this length). In section 5.4.3, we show that long-term dependencies can also be deduced from the FPT distributions. These structural informations, available in the learning sequences, are exploited in the induction algorithm POMMStruct presented in section 5.5. It starts by estimating a standard MC from the training sequences. The left part of Figure 5.6 shows an order 1 MC $T_H$ estimated from a sample containing 1000 sequences of length 100 drawn from the POMM $H$. The right part of Figure 5.5 shows the FPT distribution from `a` to `b` in the MC $T$. One can clearly observe that the FPT dynamics from state `a` to state `b` in $T_H$ poorly approximates the FPT dynamics from $\kappa_{\mathtt{a}}$ to $\kappa_{\mathtt{b}}$ in $H$ as there is only a single mode. POMMStruct iteratively adds states to the model and reestimates its probabilistic parameters in order to best match the observed FPT dynamics. Section 5.6.3 evaluates the model structure obtained by POMMStruct using data drawn from known artificial target models.

## 5.4 Modeling long-term probabilistic dependencies

This section is concerned with the modeling of probabilistic dependencies with Markovian models. The material exposed here is not a prerequisite for the induction algorithm presented in section 5.5. First, we show that standard Markov chains are not well suited to model long-term dependencies. This motivates the use of more general models like HMMs or POMMs, provided they are defined with an appropriate topology. We also show the

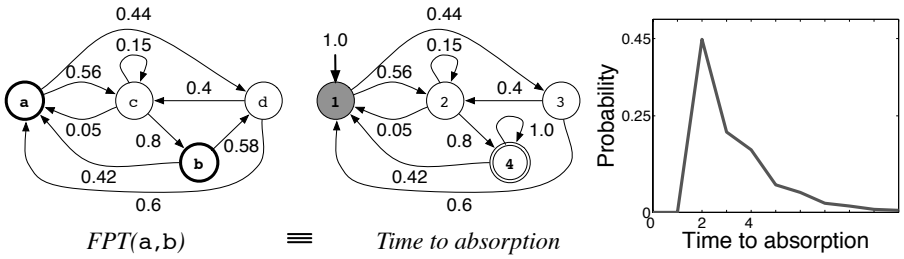



Figure 5.6: Left: An order 1 MC $T_H$ approximating the POMM $H$ displayed in the left part of Figure 5.5. Center: an absorbing MC $T$ representing the FPT from state `a` to state `b` in $T_H$ Right: The PH distribution of the absorption times in $T$, or equivalently, of the FPT between `a` to state `b` in $T_H$.

connections between long-term dependencies and the FPT dynamics.

A stochastic process $\{X_t\}_t$ contains long-term dependencies if an outcome at time $t$ significantly depends on an outcome that occurred at a much earlier time $t'$: $P[X_t|X_{t-1},\ldots,X_{t'}] \neq P[X_t|H]$ when $H = \{X_{t-1},\ldots,X_{t-p}\}$ and $p < t - t'$. Hence, the *relevant history size* for such a process is defined as the minimal size of $H$ such that $P[X_t\,|\,X_{t-1},\ldots,X_{t'}] = P[X_t\,|\,H]$, $\forall t, t' \in \mathbb{N}, t' < t$. When the size of the relevant history is bounded, Markov chains of a sufficient order can model the long-term dependencies. On the other hand, if a conditioning event $X_{t'}$ can be arbitrarily far in the past, more powerful models such as HMMs or POMMs are required. This phenomenon is further studied in section 5.4.1. Section 5.4.2 stresses the importance of the model topology in order to learn long-term dependencies with HMMs. Section 5.4.3 highlights connections between long-term dependencies and FPT.

### 5.4.1 Modeling long-term dependencies with finite order MC

Let us consider the parametric POMM $H_\theta$ displayed on the top of Figure 5.7. Emission of `e` or `f` in this model depends on whether `b` or `c` was emitted right before the last consecutive `d`'s. Depending on the number of consecutive `d`'s, the `b` or `c` outcomes can be arbitrarily far in the past. In other words, the size of the relevant history (*i.e.* the number of consecutive `d`'s + 1) is unbounded. The expected number of consecutive `d`'s is however finite and given by $\sum_{i=0}^{\infty} \theta^i = \frac{1}{1-\theta}$. Consequently, the expected size of the relevant

history is $\frac{1}{1-\theta}+1$. It should be noted that when $\theta = 0$, $H_\theta$ can be modeled accurately by an order 2 MC[3] since the relevant history size equals 2.

A model would poorly fit the distribution defined by $H_\theta$ if it would first emit `f` rather than `e` after having emitted `b`. The probability of such an event is $P_{error} = P[t_{\mathtt{f}} < t_{\mathtt{e}} \mid X_t = \mathtt{b}]$ where $t_{\mathtt{f}}$ and $t_{\mathtt{e}}$ denote the respective times of the first `f` or `e` after the outcome `b`. In the target model $H_\theta$, $P_{error} = 0$. If the same process is modeled by an order 1 MC (middle of Figure 5.7), $P_{error} = 0.5$. Indeed, when the process reaches state `d`, there is an equal probability to reach states `e` or `f`. In particular, these probabilities do not depend on previous emissions of `b` or `c`. An order 2 MC, as depicted on the bottom of Figure 5.7, would have $P_{error} = 0.475$ when $\theta = 0.95$. More generally, the error of an order $p$ MC is given by $P_{error} = \frac{\theta^{p-1}}{2}$. For instance, when $\theta = 0.95$, the expected size of the relevant history is 21 and $P_{error}$ for such a model is still 0.17. Bounding the probability of error to 0.1 would require to estimate a MC of order $p = \lceil \log_{0.95}(0.2) + 1 \rceil = 33$. An accurate estimate of such a model requires a huge amount of training data, very unlikely to be available in practice. Hence, MC offers poor approximations when the relevant history size is unbounded and more powerful models such as HMMs or POMMs are required.

### 5.4.2 Topology matters to fit long-term dependencies

Bengio has shown that the use of a good HMM topology is crucial in order to model long-term dependencies [10]. Indeed, the classical Baum-Welch algorithm applied to a fully connected graph, that is the basic technique to induce the structure via parameter estimation, is hindered by a phenomenon of diffusion of credit: the probability of being in a state at time $t$ becomes gradually independent of the states reached at a previous time $t' \ll t$. In other words, the dependencies on the past outcomes of the process end up vanishing. This phenomenon is related to the powers of the transition matrix $A$ used in the forward and backward recursions of the Baum-Welch algorithm. Let $\boldsymbol{\iota}_t$ be a row vector representing the distribution of being in each state at time $t$. This distribution $d$ steps further is given by $\boldsymbol{\iota}_{t+d} = \boldsymbol{\iota}_t A^d$. If the successive powers of $A$ converge quickly to a rank 1 matrix[4] then $\boldsymbol{\iota}_{t+d}$ becomes independent of $\boldsymbol{\iota}_t$. In such case, the estimation algorithm is likely to be stuck in an inappropriate local minimum of the likelihood.

[3] A state label `b|a` in an order 2 MC means that the process emits `b` after having emitted `a`. The probability of the transition from state `b|a` to state `d|b` encodes the second order dependence $P(X_t = d | X_{t-1} = b, X_{t-2} = a)$.

[4] All rows of a rank 1 stochastic matrix are equal.

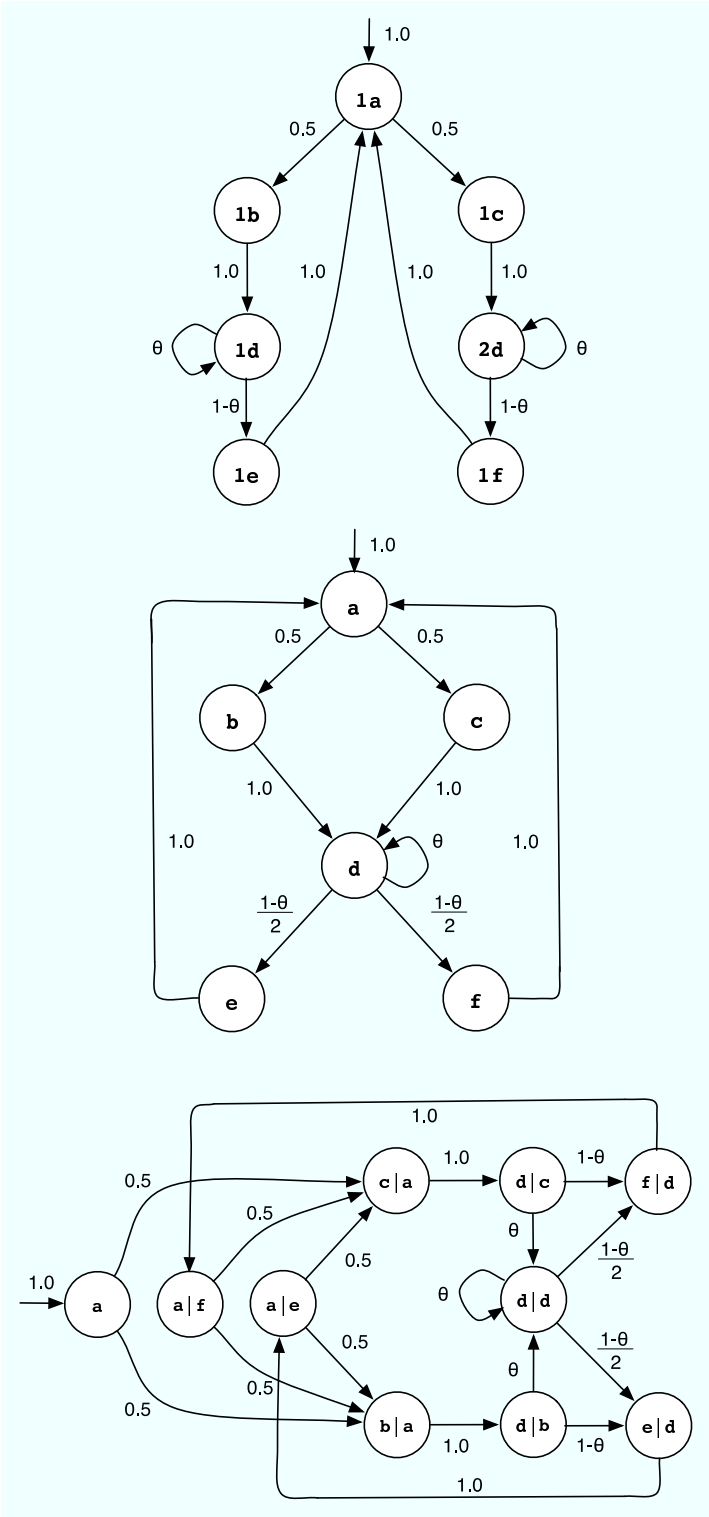



Figure 5.7: A parametric POMM $H_\theta$ (top) modeled by an order 1 MC (middle) or an order 2 MC (bottom).

For a primitive matrix $A$ (see definition 11), the rate of convergence to the rank 1 can be characterized using the Perron-Frobenius theorem (see section 1.2). It implies that a primitive stochastic matrix has a unique eigenvalue equal to 1 and that all other eigenvalues have a modulus strictly smaller than 1. If $A$ is diagonalizable and has a rank equal to $k$, then the spectral decomposition of $A$ is given by

$$A = \lambda_1 \boldsymbol{r}_1 \boldsymbol{l}_1^T + \lambda_2 \boldsymbol{r}_2 \boldsymbol{l}_2^T + \ldots + \lambda_k \boldsymbol{r}_k \boldsymbol{l}_k^T,$$

where $\lambda_i$ is the $i$-th largest eigenvalue, in absolute terms, and $\boldsymbol{r}_i$, $\boldsymbol{l}_i$ are respectively the right-hand and left-hand eigenvectors associated with $\lambda_i$. Furthermore, the spectral decomposition of $A^d$ is given by

$$A^d = \lambda_1^d \boldsymbol{r}_1 \boldsymbol{l}_1^T + \lambda_2^d \boldsymbol{r}_2 \boldsymbol{l}_2^T + \ldots + \lambda_r^d \boldsymbol{r}_k \boldsymbol{l}_k^T$$

that is, taking $A$ to the power $d$ amounts to take its eigenvalues to the power $d$. Consequently, while taking the successive powers of $A$, $\lambda_1 = 1$ remains unchanged and all other eigenvalues are decreasing until cancellation. The rate of convergence to rank 1 follows a geometric progression with a ratio equal to the absolute value of the second[5] largest eigenvalue $\lambda_2$. If $A$ is not diagonalizable (*i.e.* there are several eigenvectors associated to the same eigenvalue), the Jordan normal form [92] $A = SJS^{-1}$ is used instead of the diagonalization. One can observe that the powers of the matrix $J$ converge to a rank one matrix. The rate of convergence follows a geometric progression with a ratio upper bounded by $\lambda_2$. For more details about the convergence, see [117].

Classically, the Baum-Welch algorithm is initialized with a uniform random matrix[6]. Such a matrix is primitive and has typically a very low $\lambda_2$. The Baum-Welch algorithm is thus poorly conditioned to learn long-term dependencies when initialized in this way. On the other hand, initializing this algorithm with a matrix having $\lambda_2$ close to 1 requires prior knowledge of the model topology.

### 5.4.3 Long-term dependencies and FPT

In this section, we show that long-term dependencies between individual symbols are closely related to FPT. Let us consider the parametric POMM $H_\theta$ displayed in the top part of Figure 5.7. As mentioned in section 5.4.1, the process defined by $H_\theta$ contains long-term dependencies: the emission of `e` or `f` is conditioned by the outcome (`b` or `c`) preceding the last consecutive

[5] In the case of the POMM $T_\theta$ of Figure 5.7, $\lambda_2 = \theta$.

[6] Each entry is uniformly drawn in $[0, 1]$ and rows are normalized to sum up to 1.

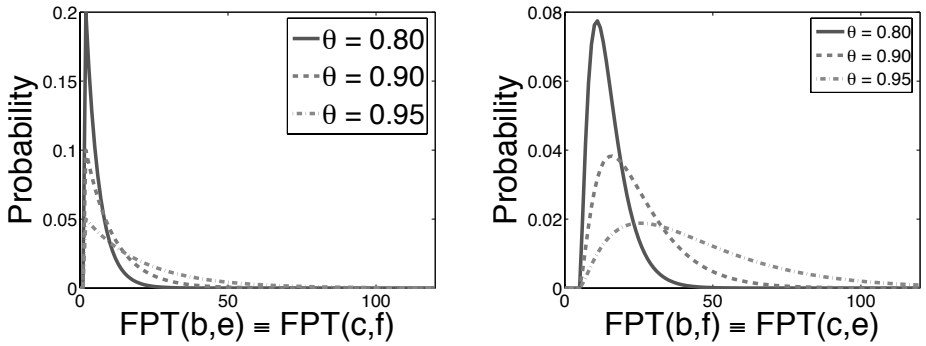



Figure 5.8: The distributions of the FPT from `b` to `e` (left) and from `b` to `f` (right) in the parametric POMM $H_\theta$ presented in Figure 5.7 for several values of $\theta$.

Table 5.1: The mean of the FPT in $H_{0.95}$ (left), modeled by an order 1 MC (center) or an order 2 MC (right).

| $H_\theta$ | e | f |
|---|---|---|
| b | 21.0 | 67.0 |
| c | 67.0 | 21.0 |

| $MC_1$ | e | f |
|---|---|---|
| b | 44.0 | 44.0 |
| c | 44.0 | 44.0 |

| $MC_2$ | e | f |
|---|---|---|
| b | 42.85 | 45.15 |
| c | 45.15 | 42.85 |

`d`'s. The $\theta$ parameter defines the length of these dependencies (*i.e.* the relevant history size). The FPT from the conditioning event (`b` or `c`) to the conditioned event (`e` or `f`) directly measures the relevant history size. Figure 5.8 displays the distributions of the considered FPT in $H_\theta$ for several values of $\theta$. One can clearly observe that increasing the value of $\theta$ gives a larger probability to long FPT. Furthermore, the plots shows that the FPT from `b` to `f` (equivalently from `c` to `e`) occur, in general, later than the FPT from `b` to `e` (equivalently from `c` to `f`) in $H_\theta$. Indeed, once the process has reached state `b`, there are at least two series of consecutive `d`'s before reaching state `f` while there is only one before reaching state `e`.

Let us now examine, the FPT distributions in the order 1 and order 2 approximating MC respectively depicted in the center and bottom parts of Figure 5.7. The considered FPT distributions for $H_\theta$ and the approximating MC are depicted Figure 5.9. The value of the $\theta$ parameter is equal to 0.95 in these plots. Table 5.1 shows the mean of these distributions. One can observe that the MC poorly approximates the dynamics of $H_\theta$ as the FPT from `b` to `e` in the approximating MC are longer than in $H_\theta$ while the FPT

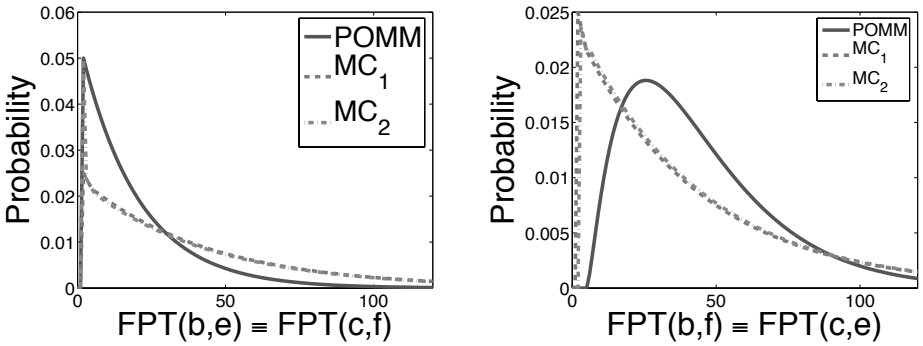



Figure 5.9: The distributions of the FPT from `b` to `e` (left) and from `b` to `f` (right) in the POMM $H_\theta$, an order 1 approximating MC and an order 2 approximating MC. The $\theta$ parameter is equal to 0.95.

from `b` to `f` are shorter than in $H_\theta$. An order 2 MC only slightly improves the fit to the target FPT with respect to an order 1 model. It should also be noted that the expectation of the relevant history size computed in section 5.4.1, equal to 21, corresponds to the mean of the FPT from `b` to `e` (or equivalently from `c` to `f`).

To sum up, the long-term dependencies between individual symbols are directly reflected in the distributions of FPT. Approximating a POMM with a finite order MC only poorly reproduces the long-term dynamics of the original process. The induction algorithm presented in section 5.5 builds a POMM to best fit the observed FPT dynamics and therefore the long-term dependencies reflected in the FPT.

## 5.5 Induction algorithm : POMMStruct

This section present the POMMStruct algorithm which learns the structure and the parameters of a POMM from a set of training sequences $S$. POMMStruct aims at inducing a model that best reproduces the FPT dynamics extracted from $S$. The objective function to be maximized is the log-likelihood of FPT observed in the training sequences (see equation (5.17)). Section 5.5.1 presents the general structure of the induction algorithm. The FPT likelihood function as well as reestimation formulas for fitting FPT distributions are detailed in section 5.5.2.

### 5.5.1 Induction scheme

The pseudo-code of POMMStruct is presented in Algorithm 3.

**Algorithm** POMMStruct
**Input**:
- A training sample $S$
- The order $r$ of the initial model (default $r = 1$)
- The number $p$ of pairs (default $p = |\Sigma|^2$)
- The number $n_r$ of restart for POMMPHit (default $n_r = 3$)
- A precision parameter $\epsilon$ (default $\epsilon = 10^{-6}$)

**Output**: A collection of POMMs

$EP_0 \leftarrow \texttt{initialize}(S, r)$;
$FPT \leftarrow \texttt{extractFPT}(S)$;
$\mathcal{P} \leftarrow \texttt{selectDivPairs}(EP, FPT, p)$;
$EP_0 \leftarrow$ POMMPHit$(EP_0, FPT, \mathcal{P}, n_r)$;
$Lik \leftarrow \texttt{FPTLikelihood}(EP_0, FPT)$;
$i \leftarrow 0$
**repeat**
  $Lik_{last} \leftarrow Lik$;
  $\kappa_{\mathtt{j}} \leftarrow \texttt{probeBlocks}(EP_i, FPT)$;
  $EP_{i+1} \leftarrow \texttt{addStateInBlock}(EP_i, \kappa_{\mathtt{j}})$;
  $EP_{i+1} \leftarrow$ POMMPHit$(EP_{i+1}, FPT, \mathcal{P}, n_r)$;
  $Lik \leftarrow \texttt{FPTLikelihood}(EP_{i+1}, FPT)$;
  $i \leftarrow i + 1$
**until** $\frac{|Lik - Lik_{last}|}{|Lik_{last}|} < \epsilon$;
**return** $\{EP_0, \ldots, EP_i\}$

Algorithm 3: POMM Induction by fitting FPT dynamics.

**1. Initialization**

Before fitting the FPT observed in the sequences, an initial model has to be defined. This operation is performed by the function `initialize`. It estimates an order $r$ MC by maximum likelihood (without smoothing) from the $N$-gram counts, as detailed in equation (1.42). In general, POMMStruct is initialized with an order 1 MC. Such a model defines the smallest POMM (in terms of number of states and transitions) that can generate the training sequences. Each block contains a single state and there is a transition from state `a` to state `b` only if the subsequence `ab` is observed. Next, the function `extractFPT` extracts the FPT in the sample for each pair of symbols according to definition 35. In section 5.3, it was shown that in general the initial MC does not adequately fit these FPT.

**2. Feature selection/weighting**

The `selectDivPairs` function is applied to determine the FPT pairs poorly fitted by the initial model. For each pair of symbols, the FPT distribution of the model is compared to the empirical FPT distribution of the sample. To do so, we propose to use the Jensen-Shannon (JS) divergence. The JS divergence [86] is a function which measures the distance between two distributions and is defined as

$$D_{JS}(P_1, P_2) = H(M) - \frac{1}{2}H(P_1) - \frac{1}{2}H(P_2) \tag{5.8}$$

where $P_1, P_2$ are two distributions over $\Omega$, $M = \frac{1}{2}(P_1 + P_2)$ and $H(P) = -\sum_{e\in\Omega} P[e] \log P[e]$ is the Shannon entropy. It can be thought of as a symmetrized and smoothed variant of the KL divergence as it is relative to the mean of the distributions. The $p$ most diverging pairs $\mathcal{P}$ are selected to be fit during the induction process, where $p$ is an input parameter. In addition, the selected pairs can be weighted according to their JS divergence in order to give more importance to the poorly fitted pairs. This is achieved by multiplying the parameters $c_i$ in FPT(`a`, `b`) (see definition 35) by the JS divergence obtained for this pair. The JS divergence is particularly well-suited for this feature weighting as it is positive and upper bounded by one [86].

**3. Iterative strategy**

At each iteration, a new state is added to the model in order to improve the fit to the observed FPT. Each iteration consists of 3 main steps: (i) determining the block in which a new state is added (ii) adding structurally the

new state in the selected block and (iii) estimating the probabilistic parameters of the augmented model. Doing so, the FPT likelihood of the training sequences monotonically increases[7] at each iteration. The algorithm is iterated until convergence of the FPT likelihood up to the precision parameter $\epsilon$.

### *3.1 Block probing*

At the beginning of each iteration, the procedure `probeBlocks` determines the block $\kappa_{\mathtt{j}}$ of the model in which a new state is added. This block is selected as the one leading to the largest FPT likelihood improvement, if any. To do so, `probeBlocks` tries successively to add a state in each block using the `addStateInBlock` procedure detailed hereafter. The parameters of the candidate model are estimated using the POMMPHIT algorithm. In order to keep things tractable, only a few, typically 20, iterations of POMMPHIT are applied to the candidate model. The block $\kappa_{\mathtt{j}}$ offering the largest improvement is returned.

### *3.2 Adding a new state*

The `addStateInBlock` function (illustrated in Figure 5.10) inserts a new state $q$ in the block $\kappa_{\mathtt{j}}$ such that $q$ is connected to all the predecessors (*i.e.* states having at least one outgoing transition to a state in $\kappa_{\mathtt{j}}$) and successors (*i.e.* states having at least one incoming transition from a state in $\kappa_{\mathtt{j}}$) of $\kappa_{\mathtt{j}}$. These two sets need not to be disjoint and may include states in $\kappa_{\mathtt{j}}$ (if they are connected to some state(s) in $\kappa_{\mathtt{j}}$). This point is illustrated in Figure 5.10. The transition probabilities of the augmented model are (re)initialized as follows. For all predecessors $p_i$ of $\kappa_{\mathtt{j}}$, the outgoing transition probabilities are:

$$A'_{p_iq'} = \begin{cases} A_{p_iq'} & \text{if } q' \in Q \setminus \kappa_{\mathtt{j}} \\ rA_{p_iq'}/(r+1) & \text{if } q' \in \kappa_{\mathtt{j}} \setminus q \\ \sum_{q' \in \ \kappa_{\mathtt{j}}} A_{p_iq'}/(r+1) & \text{if } q' = q \end{cases} \tag{5.9}$$

where $r$ is the number transitions from $p_i$ to $\kappa_j \setminus q$. That is, the probabilities of the existing transitions leading to $\kappa_j$ (the dotted transitions in Figure 5.10) are uniformly discounted and the discounted mass is assigned to the new transition $p_i \rightarrow q$ (the bold transitions in Figure 5.10). The probabilities of the transitions originated from the new state $q$ (the dashed transitions in

[7] When a new state is added, some of the additional transitions are randomly initialized (see Figure 5.10). Doing so, the likelihood of the augmented model can temporarily decrease. However, in most cases, reestimating these parameters with POMMPHIT yields a likelihood increase with respect to the model before state addition. Otherwise, the stopping criterion of POMMSTRUCT is triggered.

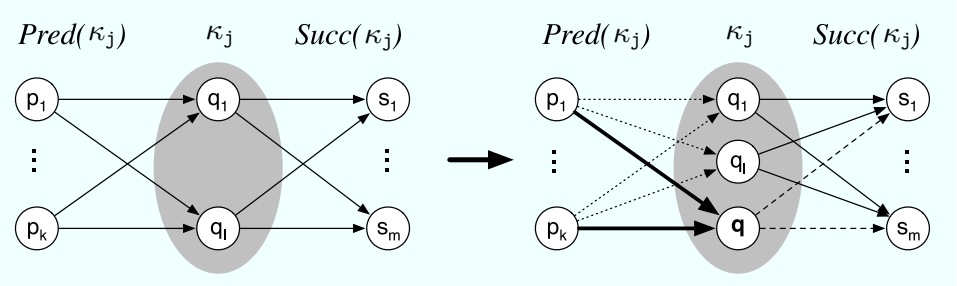



Figure 5.10: Left: The block $\kappa_j$ in a POMM with its predecessors and successors. Right: The same block after having added the new state $q$. The dashed transitions are randomly initialized in POMMPHIT. The dotted transitions are uniformly discounted and the subtracted probability mass is assigned to the new transitions leading to $q$ displayed in bold. Plain line transitions are initialized to their current value.

Figure 5.10) are randomly initialized in POMMPHIT. All other transition probabilities of the model remain unchanged (the plain line transitions in Figure 5.10). Similarly, the initial probabilities $\boldsymbol{\sigma}'^{\kappa_j}$ for the FPT originated in $\kappa_j$ are initialized using an uniform discounting:

$$\sigma'^{\kappa_j}_{q'} = \begin{cases} r\sigma^{\kappa_j}_{q'}/(r+1) & \text{if } q' \in \kappa_\mathtt{j} \setminus q \\ 1/(r+1) & \text{if } q' = q \\ 0 & \text{otherwise} \end{cases} \tag{5.10}$$

where $r$ is the number of states in $\kappa_j$ before adding the new state $q$.

*3.3 Parameter estimation and trimming*

The probabilistic parameters of the augmented model are estimated using POMMPHIT (see section 5.5.2) until convergence. POMMPHIT is first applied to the transitions connected to $\kappa_\mathtt{j}$, *i.e.* the transitions depicted in the right part of Figure 5.10. This procedure is called *local estimation*. POMMPHIT is restarted $n_r$ times with different random initializations of the dashed transitions ($n_r$ is an input parameter) and the parameters offering the largest FPT likelihood are kept. Starting from these estimated parameters, POMMPHIT is then applied to all the model parameters. This procedure is called *global estimation*. The local estimation is intended to provide a good starting point for the global estimation in a reasonable time. An interesting byproduct of POMMPHIT are the expected transition passage times (see section 5.5.2). It provides the average number of times the transitions are triggered when observing the FPT in the sample. According

to this criterion, the less frequently used transitions (determined at convergence) are successively trimmed off from the model. Whenever a transition is removed, the model is normalized to insure properness constraints and the parameters are reestimated using POMMPHit. In general, the convergence is attained after a few iterations as the parameters not affected by the trimming are already well estimated. Transitions are trimmed until the FPT likelihood no longer increases. This procedure has several benefits: (i) it can move POMMPHit away from a local minimum of the FPT likelihood function (ii) it makes the model sparser and therefore reduces the computational resources needed in the forward-backward computations (see section 5.5.2) and (iii) the obtained model is more interpretable. The trimming procedure is applied after both the local and global estimations.

**4. Selecting the optimal model size**

The present discussion is about *model selection* and more precisely about the selection of the optimal model size. When the FPT likelihood has converged up to a user-defined precision parameter $\epsilon$, POMMStruct returns *all the models* $\{EP_0, \ldots, EP_i\}$ considered during induction from the initial MC to the last estimated POMM. Each model is evaluated on an independent validation set of sequences and the model offering the highest FPT likelihood is chosen. It is important to note that the selected model size does not depend much on the precision parameter $\epsilon$ as all the models returned by POMMStruct are considered. In our experiments, we set $\epsilon = 10^{-6}$.

At each iteration, the computational complexity is dominated by the complexity of POMMPHit (see section 5.5.2). POMMStruct does not maximize the likelihood of the training sequences in the model but the likelihood of the FPT extracted from these sequences. We argued in section 5.3 that maximizing this criterion is relevant to learn an adequate model topology. If one wants to perform sequence prediction, *i.e.* predicting the next outcomes of a process given its past history, the parameters of the model may be adjusted towards this objective. This can be achieved by applying the standard Baum-Welch procedure initialized with the model resulting from POMMStruct.

Finally, let us mention that POMMStruct is not guaranteed to return a canonical POMM, *i.e.* an equivalent model with fewer states or transitions may exist. The question of reducing a HMM (let us recall that POMMs form a particular case of HMMs) to a canonical form is thoroughly studied in [7]. An algorithm is proposed to reduce a HMM to a Generalized Markov Model (GMM), *i.e.* a HMM relaxed such that its numerical parameters

belong to $\mathbb{R}$ rather than to $[0, 1]$. The reduced model is minimal in term of number of states. Such a technique could be applied to the POMMs resulting from POMMStruct. It should be pointed out that the obtained canonical GMMs are however not necessarily easier to interpret than the original POMMs as they do not always have a probabilistic interpretation. Indeed, such models may contain numerical parameters greater than one or negative parameters which have to be interpreted as inhibitions to perform some events (a transition or a symbol emission).

### 5.5.2 Fitting the FPT: POMMPHit

In this section, we introduce the POMMPHit algorithm for fitting the FPT distributions between blocks in a POMM $H = \langle \Sigma, Q, A, B, \iota \rangle$ from the FPT observed in sequences. POMMPHit is based on the Expectation-Maximization (EM) algorithm. In the context of sequence classification (see section 7.3.1), a simplified version of POMMPHit, called PHit, is presented. PHit only fits a single PH distribution using an absorbing MC whereas POMMPHit fits several PH distributions using a POMM. For each pair of symbol $(\mathtt{a}, \mathtt{b})$, the observations consist of the FPT $Z = \{(z_1, c_1), \ldots, (z_l, c_l)\}$ extracted from the sequences according to definition 35.

POMMPHit makes two modeling assumptions to fit the observed FPT. First, the observations for a given pair $(\mathtt{a}, \mathtt{b})$ are assumed to be independent from the observations associated to the other pairs. While this assumption is generally not satisfied, it drastically simplifies the reestimation formula and consequently offers an important computational speed-up. Moreover, good results are obtained in practice. Secondly, the starting distributions $\boldsymbol{\sigma}^{\kappa_x}$ of the FPT from each block $\kappa_x$ in the model are considered as additional model parameters to be estimated from the sample rather than deduced from the other parameters.

A passage time $z_i$ is considered here as an incomplete observation of the pair $(z_i, h_i)$ where $h_i$ is the hidden path (*i.e.* the sequence of states) taken by the process to go from block $\kappa_{\mathtt{a}}$ to block $\kappa_{\mathtt{b}}$ in $z_i$ steps. In the sequel, $\mathcal{H}^{\mathtt{a},\mathtt{b}}$ denotes the set of hidden paths from block $\kappa_{\mathtt{a}}$ to block $\kappa_{\mathtt{b}}$. EM finds the maximum likelihood estimates of the parameters $\rho$ of a joint distribution $P(Z, \mathcal{H} \mid \rho)$ when the variable $\mathcal{H}$ is unobserved ($\mathcal{H}$ is a *latent* or *hidden* variable). This is achieved by iteratively computing the parameters $\rho$ that maximize $\mathbb{E}\left[\ln(P(Z, \mathcal{H} \mid \rho)) \mid Z\right]$. Each EM iteration involves two steps: (i) the computation of the conditional expectation of the latent variables given the last parameter values $\rho^{t-1}$ and the partial observations $Z$, (ii) the maximization of the parameters $\rho^t$ given the conditional expectation of

the latent variables. Before presenting the expectation and maximization steps in POMMPHit, let us introduce auxiliary hidden variables which provide sufficient statistics to compute the complete FPT likelihood function $P[Z, \mathcal{H} \mid \rho]$ conditioned to the model parameters $\rho$:

- $S^{\mathtt{a},\mathtt{b}}(q)$: the number of observations in $\mathcal{H}^{\mathtt{a},\mathtt{b}}$ starting in state $q \in \kappa_{\mathtt{a}}$,
- $N^{\mathtt{a},\mathtt{b}}(q, q')$: the number of times state $q'$ immediately follows state $q$ in $\mathcal{H}^{\mathtt{a},\mathtt{b}}$.

The complete FPT likelihood function is defined as follows:

$$P[Z, \mathcal{H} \mid \rho] = \prod_{(\mathtt{a},\mathtt{b}) \in \mathcal{P}} \prod_{q \in \kappa_{\mathtt{a}}} (\sigma_q^{\kappa_{\mathtt{a}}})^{S^{\mathtt{a},\mathtt{b}}(q)} \prod_{q,q' \in Q} A_{qq'}^{N^{\mathtt{a},\mathtt{b}}(q,q')} \tag{5.11}$$

where $\boldsymbol{\sigma}^{\kappa_{\mathtt{a}}}$ is the initial distribution over $\kappa_{\mathtt{a}}$ for the FPT starting in $\kappa_{\mathtt{a}}$.

**Expectation step**

The expectations of the variables $S^{\mathtt{a},\mathtt{b}}(q)$ and $N^{\mathtt{a},\mathtt{b}}(q, q')$ are conveniently computed using *forward* and *backward* variables similar to those used in the Baum-Welch algorithm (see section 2.3.1). It is assumed here that $\mathtt{a} \neq \mathtt{b}$. Otherwise, the states in $\kappa_{\mathtt{a}}$ have to be duplicated in order to have distinct starting and destination states. Given a state $q \in Q$ and a time $t \in \mathbb{N}$, the forward variable $\alpha^{\mathtt{a},\mathtt{b}}(q, t)$ computes the probability that the process started in $\kappa_{\mathtt{a}}$ reaches state $q$ after $t$ steps without transiting through states in $\kappa_{\mathtt{b}}$:

$$\alpha^{\mathtt{a},\mathtt{b}}(q, t) = P[X_t = q, \{X_k\}_{k=1}^{t-1} \in Q \setminus \kappa_{\mathtt{b}} \mid X_0 \in \kappa_{\mathtt{a}}] \tag{5.12}$$

The forward variables can be computed using the following recurrence for $q \in Q$ and $t \in \mathbb{N}$:

$$\begin{array}{lrcl} \text{(Base case)} & \alpha^{\mathtt{a},\mathtt{b}}(q, 0) & = & \begin{cases} \sigma_q^{\kappa_{\mathtt{a}}} & \text{if } q \in \kappa_{\mathtt{a}} \\ 0 & \text{otherwise} \end{cases} \\ \text{(Induction)} & \alpha^{\mathtt{a},\mathtt{b}}(q, t) & = & \sum_{q' \in Q \setminus \kappa_{\mathtt{b}}} \alpha^{\mathtt{a},\mathtt{b}}(q', t-1) A_{q'q} \end{array} \tag{5.13}$$

where $\boldsymbol{\sigma}^{\kappa_{\mathtt{a}}}$ denotes the initial distribution for the FPT originated in $\kappa_{\mathtt{a}}$. Given a state $q \in Q$ and a time $t \in \mathbb{N}$, the backward variable $\beta^{\mathtt{b}}(q, t)$ computes the probability that state $q$ is reached by the process $t$ steps before reaching $\kappa_{\mathtt{b}}$ for the first time:

$$\beta^{\mathtt{b}}(q, t) = P[X_0 = q, \{X_k\}_{k=1}^{t-1} \in Q \setminus \kappa_{\mathtt{b}} \mid X_t \in \kappa_{\mathtt{b}}] \tag{5.14}$$

The following recurrence computes the backward variables for $q \in Q$ and $t \in \mathbb{N}$:

$$\begin{aligned}
\text{(Base case)} \quad \beta^{\mathtt{b}}(q,0) &= \begin{cases} 1 & \text{if } q \in Q_A \\ 0 & \text{otherwise} \end{cases} \\
\text{(Induction)} \quad \beta^{\mathtt{b}}(q,t) &= \begin{cases} 0 & \text{if } q \in Q_A \\ \sum_{q' \in Q} \beta^{\mathtt{b}}(q', t-1) A_{qq'} & \text{otherwise} \end{cases}
\end{aligned} \tag{5.15}$$

The forward and backward recurrences are efficiently computed using a $|Q| \times L^{\mathtt{a},\mathtt{b}}$ lattice structure where $L^{\mathtt{a},\mathtt{b}}$ is the longest observed FPT from $\mathtt{a}$ to $\mathtt{b}$. The probability distribution of the FPT from block $\kappa_{\mathtt{a}}$ to block $\kappa_{\mathtt{b}}$ can be computed as follows with the forward or backward recurrences:

$$P[\text{fpt}(\mathtt{a},\mathtt{b}) = k] \propto \sum_{q \in \kappa_{\mathtt{b}}} \alpha^{\mathtt{a},\mathtt{b}}(q,k) = \sum_{q \in \kappa_{\mathtt{a}}} \sigma^{\mathtt{a}}_q \beta^{\mathtt{b}}(q,k) \;\; \forall k \in \mathbb{N}^0 \tag{5.16}$$

Consequently, the log-likelihood of the observed FPT given the model parameters $\rho$ is provided by

$$\begin{aligned}
\log(P[Z \mid \rho]) &= \sum_{(\mathtt{a},\mathtt{b}) \in \mathcal{P}} \sum_{(z,c) \in \text{FPT}(\mathtt{a},\mathtt{b})} c \, \log \left( \sum_{q \in \kappa_{\mathtt{b}}} \alpha^{\mathtt{a},\mathtt{b}}(q,z) \right) \\
&= \sum_{(\mathtt{a},\mathtt{b}) \in \mathcal{P}} \sum_{(z,c) \in \text{FPT}(\mathtt{a},\mathtt{b})} c \, \log \left( \sum_{q \in \kappa_{\mathtt{a}}} \sigma^{\mathtt{a}}_q \beta^{\mathtt{b}}(q,z) \right)
\end{aligned} \tag{5.17}$$

**Conditional expectation of** $S^{\mathtt{a},\mathtt{b}}(q)$: The auxiliary variable $S^{\mathtt{a},\mathtt{b}}(q)$ can be decomposed as a sum of indicator variables: $S^{\mathtt{a},\mathtt{b}}(q) = \sum_{k=1}^{l} \mathbb{1}\{h_{k,0} = q\}$ where the notation $h_{k,j}$ denotes the $j$-th state in the $k$-th hidden path from $\kappa_{\mathtt{a}}$ to $\kappa_{\mathtt{b}}$. The conditional expectation $\overline{S^{\mathtt{a},\mathtt{b}}}(q) = \mathbb{E}[S^{\mathtt{a},\mathtt{b}}(q) | (z_1,c_1),\ldots,(z_l,c_l)]$ is given by

$$\begin{aligned}
\overline{S^{\mathtt{a},\mathtt{b}}}(q) &= \mathbb{E}\left[ \sum_{k=1}^{l} \mathbb{1}\{h_{k,0} = q\} \,\middle|\, (z_1,c_1),\ldots,(z_l,c_l) \right] \\
&= \sum_{k=1}^{l} c_k \mathbb{E}\left[ \mathbb{1}\{h_{k,0} = q\} \mid z_k \right] \\
&= \sum_{k=1}^{l} c_k P[h_{k,0} = q \mid z_k] = \sum_{k=1}^{l} c_k \frac{P[h_{k,0} = q, z_k]}{P[z_k]}
\end{aligned} \tag{5.18}$$

Using equation (5.16), $P[z_k] = \sum_{q \in \kappa_{\mathtt{b}}} \alpha^{\mathtt{a},\mathtt{b}}(q,z_k)$ while $P[h_{k,0} = q, z_k]$ is readily obtained using the backward variables: $P[h_{k,0} = q, z_k] = \sigma^{\mathtt{a}}_q \beta^{\mathtt{b}}(q,z_k)$. Finally, the desired conditional expectation is defined as follows:

$$\overline{S^{\mathtt{a},\mathtt{b}}}(q) = \sum_{k=1}^{l} c_k \frac{\sigma^{\mathtt{a}}_q \beta^{\mathtt{b}}(q,z_k)}{\sum_{q \in \kappa_{\mathtt{b}}} \alpha^{\mathtt{a},\mathtt{b}}(q,z_k)} \tag{5.19}$$

**Conditional expectation of** $N^{\mathtt{a},\mathtt{b}}(q,q')$: The auxiliary variable $N^{\mathtt{a},\mathtt{b}}(q,q')$ can be decomposed as $\sum_{k=1}^{l}\sum_{t=0}^{z_k-1}\mathbb{1}\{h_{k,t}=q,h_{k,t+1}=q'\}$, that is, counting the presence of the transition $q\to q'$ in each hidden path from $\kappa_{\mathtt{a}}$ to $\kappa_{\mathtt{b}}$. The conditional expectation $\overline{N^{\mathtt{a},\mathtt{b}}}(q,q')=\mathbb{E}\left[N^{\mathtt{a},\mathtt{b}}(q,q')\mid(z_1,c_1),\ldots,(z_l,c_l)\right]$ is provided by

$$\begin{aligned}\overline{N^{\mathtt{a},\mathtt{b}}}(q,q') &= \mathbb{E}\left[\sum_{k=1}^{l}\sum_{t=0}^{z_k-1}\mathbb{1}\{h_{k,t}=q,h_{k,t+1}=q'\}\,\middle|\,(z_1,c_1),\ldots,(z_l,c_l)\right]\\ &= \sum_{k=1}^{l}c_k\sum_{t=0}^{z_k-1}\frac{P[h_{k,t}=q,h_{k,t+1}=q',z_k]}{P[z_k]}\end{aligned} \tag{5.20}$$

The probability $P[h_{k,t}=q,h_{k,t+1}=q',z_k]$ can be computed as follows

$$P[h_{k,t}=q,h_{k,t+1}=q',z_k]=\alpha^{\mathtt{a},\mathtt{b}}(q,t)A_{qq'}\beta^{\mathtt{b}}(q',z-t-1) \tag{5.21}$$

That is, the probability to reach state $q$ at time $t$, to perform the transition $q\to q'$, and to reach $\kappa_{\mathtt{b}}$ for the first time $z-t-1$ steps later. Finally, the desired expectation is provided by

$$\overline{N^{\mathtt{a},\mathtt{b}}}(q,q')=\sum_{k=1}^{l}c_k\sum_{t=0}^{z_k-1}\frac{\alpha^{\mathtt{a},\mathtt{b}}(q,t)A_{qq'}\beta^{\mathtt{b}}(q',z_k-t-1)}{\sum_{q\in\kappa_{\mathtt{b}}}\alpha^{\mathtt{a},\mathtt{b}}(q,z_k)} \tag{5.22}$$

The conditional expectations $\overline{S^{\mathtt{a},\mathtt{b}}}(q)$ and $\overline{N^{\mathtt{a},\mathtt{b}}}(q,q')$ are used in the maximization step of POMMPHit but also in the trimming procedure of POMMStruct. In particular, $\sum_{(\mathtt{a},\mathtt{b})\in\mathcal{P}}\overline{N^{\mathtt{a},\mathtt{b}}}(q,q')$ provides the average number of times the transition $q\to q'$ is triggered while observing the sample FPT.

### Maximization step

Given the conditional expectations $\overline{S^{\mathtt{a},\mathtt{b}}}(q)$ and $\overline{N^{\mathtt{a},\mathtt{b}}}(q,q')$, the maximum likelihood estimates of the POMM parameters are the following for all $q,q'\in Q$:

$$\sigma_q^{\kappa_{\mathtt{a}}} = \begin{cases}\dfrac{\sum_{\mathtt{b}\in\{\mathtt{b}|(\mathtt{a},\mathtt{b})\in\mathcal{P}\}}\overline{S^{\mathtt{a},\mathtt{b}}}(q)}{\sum_{q\in\kappa_{\mathtt{a}}}\sum_{\mathtt{b}\in\{\mathtt{b}|(\mathtt{a},\mathtt{b})\in\mathcal{P}\}}\overline{S^{\mathtt{a},\mathtt{b}}}(q)} & \text{if } q\in\kappa_{\mathtt{a}}\\ 0 & \text{otherwise}\end{cases} \tag{5.23}$$

$$A_{qq'} = \frac{\sum_{(\mathtt{a},\mathtt{b})\in\mathcal{P}}\overline{N^{\mathtt{a},\mathtt{b}}}(q,q')}{\sum_{q'\in Q}\sum_{(\mathtt{a},\mathtt{b})\in\mathcal{P}}\overline{N^{\mathtt{a},\mathtt{b}}}(q,q')} \tag{5.24}$$

The computational complexity per iteration is $\Theta(pL^2n_t)$ where $p$ is the number of selected pairs, $L$ is the longest observed FPT and $n_t$ is the number of

transitions in the current model. An equivalent bound for this computation is $\mathcal{O}(pL^2|Q|^2)$, but this upper bound is tight only if the transition matrix $A$ is dense.

## 5.6 Experiments

This section presents experimental results obtained with POMMStruct on several sequence modeling tasks. The objectives of this experimental study are the following:

1. to evaluate the quality of the model structure learned with POMMStruct,
2. to evaluate the ability of POMMStruct to model processes having a complex FPT dynamics including long-term dependencies (see section 5.4),
3. to observe the influence of the hyperparameters on the estimated models and on the CPU time taken to learn these models,
4. to compare the performance of POMMStruct with respect to standard approaches to the structural induction of HMMs: (i) Baum-Welch applied on fully-connected graphs with transition trimming (see 4.1.1) and the Bayesian state merging algorithm due to Stolcke (see section 4.2.2).

### 5.6.1 Datasets

This section presents the datasets used in our experiments. The first, called `GenDep`, consists of artificially generated sequences drawn from target POMMs having a complex FPT dynamics including long-term dependencies. The second dataset is a part of the `Splice` dataset also used in sequence classification (see section 7.5.1). The third dataset, called `CopChrom` consists of sequences encoding chromosome samples.

#### Artificially generated data: `GenDep`

In order to assess the performances of the proposed approach in a controlled setting, a set of target models with a complex FPT dynamics was randomly generated. The `GenDep` dataset is made of sequences drawn from these models. Target models are created by queuing sequences of 4 states, called here *branches* (see Figure 5.11), in a cyclic fashion. The first and the third states of a branch are called *specific states*, as they emit a specific symbol, not emitted by any other state in the model. The second and the fourth

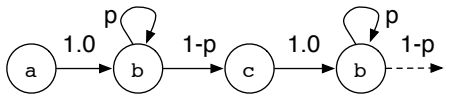



Figure 5.11: A branch in the artificial target POMMs.

states of a branch, called *forgetting states*, emit the same letter and have a self loop with a random probability $p \geq 0.7$. These states generate long-term dependencies between specific states as the process remains for several steps (in average $\frac{1}{1-p}$) in forgetting states. For instance, in the branch displayed in Figure 5.11, a large number of consecutive `b`'s are emitted and the symbols directly following these `b`'s depends on the symbol (`a` or `c`) emitted right before these `b`'s. The symbols emitted in a branch are not used elsewhere and, therefore, the relation between the alphabet size and the number of states is here $|\Sigma| = \frac{3}{4}|Q|$. Afterward, $2|Q|$ random transitions of small magnitude $\leq 0.1$ are added[8] to the model in order to make the structure more general while preserving its complex FPT dynamics. We generated target models containing 4, 8, 16, 24 and 32 states. From each target model, 500 training sequences and 250 test sequences of length 100 were generated.

### Splice

The `Splice` dataset, available from the UCI repository, is made of DNA sequences, each one consisting of 60 nucleotides. A more detailed description of this dataset is provided in section 7.5.1. We focus here on sequences containing "exon ⇒ intron" boundaries. Furthermore, the rare sequences containing ambiguities have been filtered out to limit the alphabet to the 4 nucleotides `A,C,T,G`. Note that our method can cope with ambiguous sequences; they were not included here due to their low frequency in the dataset (less than 0.5% of sequences contain ambiguities). The considered training and test sets contain respectively 500 and 235 sequences.

### CopChrom

The `CopChrom` database[9] consists of a collection of chromosome sequences classified into 22 categories (or types). Each database entry is made of (i) the metaphase from which the chromosome sample is extracted [1..180], (ii)

[8] When a random transition is added, the model is normalized to ensure that the total outgoing probability mass from each state equals 1.

[9] The `CopChrom` dataset is publicly available at the address:
http://algoval.essex.ac.uk:8080/data/sequence/copchrom/

the chromosome type [0..21], the centromere position and (iii) a sequence defined on an alphabet consisting of the lower case letters `a-f`, the uppercase letters `A-F` and the symbol '='. In our experiments, all the informations but the sequences have been filtered out. A letter codes a difference of successive sample positions whereas '=' means that two successive positions are equal. The upper and lower cases are respectively used to represent positive and negative differences. The alphabetic position codes the level of the difference, *e.g.* `a` stands for a negative difference of 1 whereas `E` denotes a positive difference of 5. For each of the 22 categories, the database contains 100 training and test sequences. The training and test sequences have been gathered into sets of 200 sequences and performances are assessed for each class using a standard tenfold cross-validation.

### 5.6.2 Evaluation criteria

The dynamics and the prediction performances of the models are respectively assessed using (i) the *FPT divergence* and (ii) the *perplexity* which are detailed hereafter. Other performance measures to assess the model structure are used in subsection 5.6.3. To evaluate the FPT dynamics of a model $H$, we propose to compute the JS divergence between the PH distributions defined by $H$ and the empirical FPT distributions observed in a test sample $S_{\text{test}}$. The JS divergence has already been used in the feature selection procedure presented in section 5.5.1. It is averaged here over all the symbol pairs $(\mathtt{a}, \mathtt{b})$ with $\mathtt{a}, \mathtt{b} \in \Sigma$ such that each pair is weighted by its relative frequency in the test sample $S_{\text{test}}$:

$$\text{Div}_{\text{FPT}}(H, S_{\text{test}}) = \sum_{\mathtt{a},\mathtt{b} \in \Sigma} \frac{\text{count}(\mathtt{a}, \mathtt{b})\ D_{JS}(\widehat{\text{FPT}}_{S_{\text{test}}}(\mathtt{a}, \mathtt{b}), P[\text{fpt}(\mathtt{a}, \mathtt{b}) \mid H])}{\sum_{\mathtt{a}',\mathtt{b}' \in \Sigma} \text{count}(\mathtt{a}', \mathtt{b}')} \tag{5.25}$$

where $\text{count}(\mathtt{a}, \mathtt{b})$ is the cumulated frequency of the FPT from `a` to `b` in $S_{\text{test}}$, $\widehat{\text{FPT}}_{S_{\text{test}}}(\mathtt{a}, \mathtt{b})$ denotes the empirical distribution of FPT from `a` to `b` in $S_{\text{test}}$ and P[fpt(`a`,`b`) | H] denotes the PH distribution from `a` to `b` defined by the model $H$.

The perplexity (PP) is related to the sequence likelihood and is used to measure the prediction efficiency of probabilistic models. It is formally defined as follows:

$$PP = 2^{-\frac{1}{||S_{\text{test}}||} \sum_{s \in S_{\text{test}}} \log_2 P[s \mid H]} \tag{5.26}$$

where $||S_{\text{test}}||$ denotes the total number of symbols in the test sample $S_{\text{test}}$ and $P[s \mid H]$ denotes the probability of generating the sequence $s$ from the

model $H$. The perplexity can be interpreted as a measure of the average uncertainty for predicting the next symbol at any given position in the test strings. An uninformed model would predict any symbol of the alphabet $\Sigma$ with the same probability $1/|\Sigma|$. Such a model would have $PP = |\Sigma|$. There is a potential issue when computing the perplexity on test sequences. Indeed, the estimated model may give a null probability to these sequences, yielding an infinite perplexity. To face this problem, the emission distributions of states are smoothed such that each symbol of the alphabet is given a minimal probability of $10^{-5}$.

### 5.6.3 Evaluation of the model structure

Sections 5.3 and 5.4 argue that using a good model structure or topopology is crucial to accurately model the FPT dynamics of a process. In turn, one may wonder whether inducing a model fitting the observed FPT provides a good model structure. This is precisely the objective of the current section. The quality of the model structure induced by POMMStruct is evaluated here in a controlled setting. To do so, the artificial models used to generate the sequences of the `GenDep` dataset (see section 5.6.1) are considered as reference target models. For each target size (*i.e.* 4, 8, 16, 24 and 32 states), the complete training set (*i.e.* 500 sequences) is used to estimate a model with POMMStruct. The induced models are then structurally compared with the target models. For comparison purpose, POMMStruct is stopped when the target size (in terms of number of states) is reached and the current model is returned (*i.e.* there is no model size selection using a validation set like in section 5.5.1). Comparisons with target models are performed using the two following criteria:

$$\text{MissAdd}(Tar, EP) = \frac{1}{n_s(Tar)} \sum_{\mathtt{a} \in \Sigma} \max(n_s(EP, \mathtt{a}) - n_s(Tar, \mathtt{a}), 0) \quad (5.27)$$

$$\Delta\text{Trans}(Tar, EP) = \frac{1}{n_t(Tar)} |n_t(EP) - n_t(Tar)| \quad (5.28)$$

where $Tar$ and $EP$ respectively denote the target and the estimated model, $n_s(H, \mathtt{a})$ denotes the number of states emitting "`a`" in a POMM $H$, and $n_s(H)$ and $n_t(H)$ are respectively the number of states and the number of transitions with a strictly positive probability in a POMM $H$.

On the one hand, MissAdd($Tar, EP$) is the proportion of states wrongly added in a block relatively to the number of states in the target. In other words, it is the proportion of *false positive* state additions. The left part of Figure 5.12 evaluates this criterion for increasing target model sizes. For target models up to 16 states, POMMStruct makes no faulty state additions to the model. For targets having 24 and 32 states, POMMStruct wrongly adds two states to the model and the relative scores are respectively $2/24 = 8.33\%$ and $2/32 = 6.25\%$.

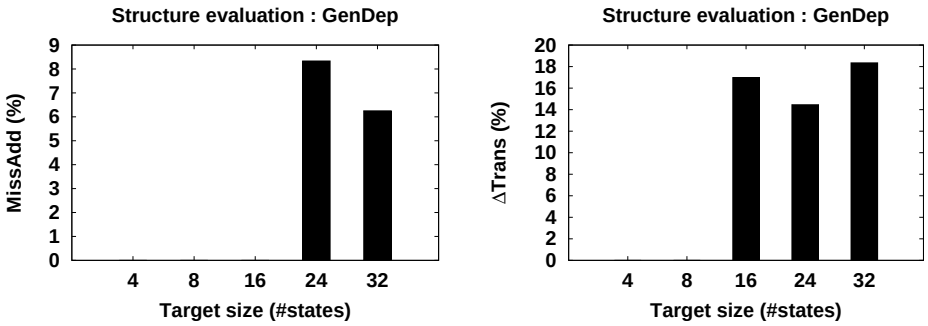



Figure 5.12: Evaluation of the estimated model structure induced by POMMStruct using the `GenDep` dataset and associated target models. Left: evaluation of the proportion of states added in a wrong block for increasing target model sizes. Right: evaluation of the relative difference of number of transitions between the target and the estimated model for increasing target model sizes.

On the other hand, $\Delta\text{Trans}(Tar, EP)$ is the difference, in absolute value, between the number of transitions in the target and in the estimated model, normalized by the number of transitions in the target. This criterion is assessed in the right part of Figure 5.12. For target sizes up to 8 states, the target and the induced model have exactly the same number of transitions. Furthermore, when considering a target size of 4, the estimated model and the target are isomorphic (this is not the case for a target model with 8 states). For larger target sizes, POMMStruct tends to estimate models with more transitions than in the target. For instance, for a target size of 32 states, the estimated model has 129 transitions whereas the target has only 109 transitions which result on a relative difference of $(129 - 109)/109 = 18.35\%$.

### 5.6.4 Complex dynamics modeling

In this section, we visually assess the ability of POMMStruct to model complex FPT dynamics. To do so, POMMStruct is applied to sequences from the `GenDep` dataset. These sequences are drawn from artificially generated target POMMs which include a broad range of dynamics and long-term dependencies. Figure 5.13 displays selected PH distributions defined by a target POMM $H$ defined on an alphabet of 24 symbols and containing 32 states. It also shows empirical FPT distributions of a learning sample drawn from $H$ and containing 500 sequences as well as the PH distributions of a model estimated with POMMStruct. The concerned symbol pair as well

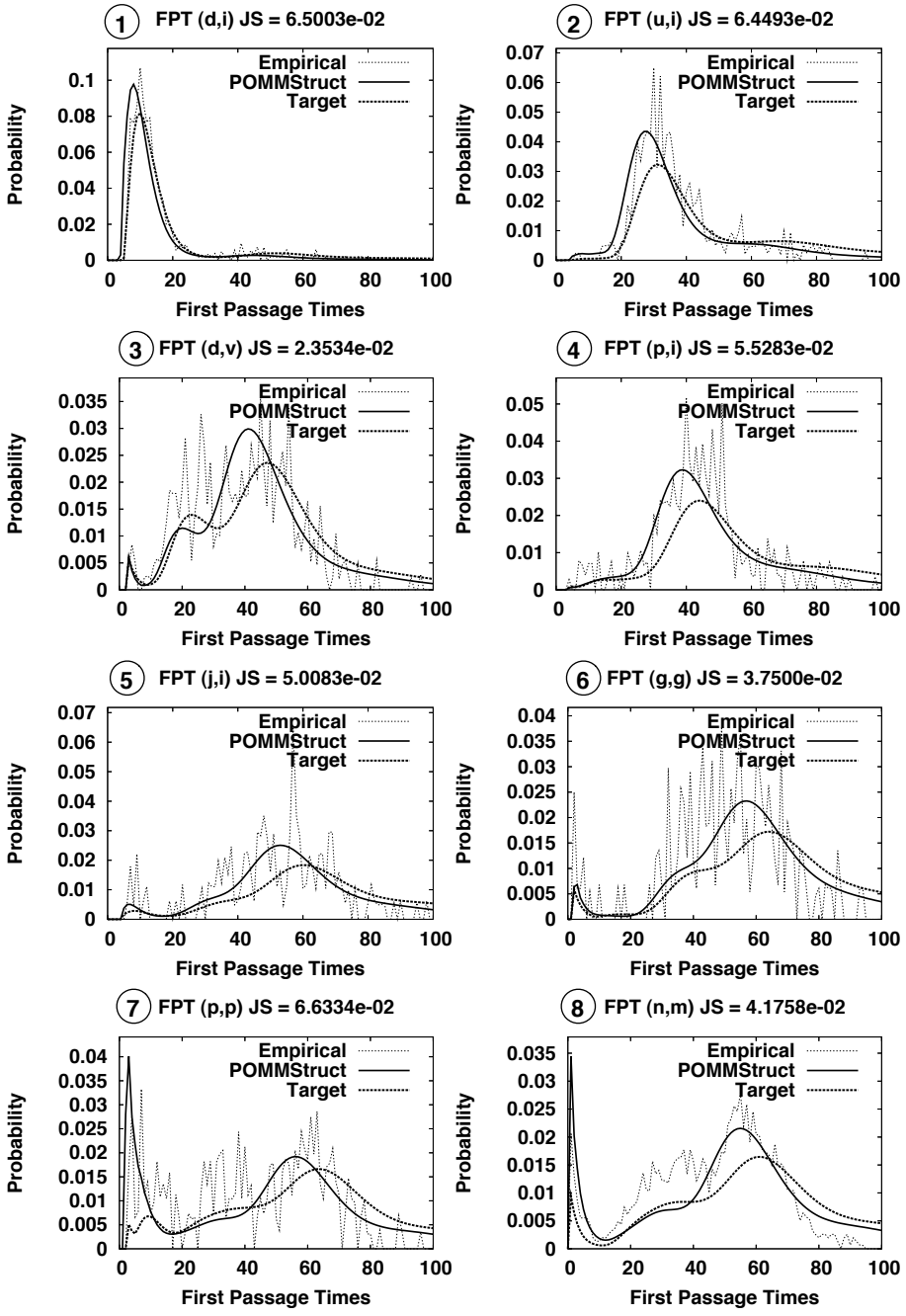



Figure 5.13: Selected FPT fits obtained with POMMStruct applied to the `GenDep` dataset. The solid line represents the PH distribution of the estimated model. The light dashed line denotes the empirical FPT observed in the 500 training sequences. The bold dashed line stands for the PH distribution of the target model.

as the Jensen-Shannon divergence between the target and the estimated PH distribution are shown above each plot. Target PH distributions 1 and 2 are respectively shaped like hypergeometric and generalized binomial distributions. The target distribution 2 is shifted to the right as FPT shorter than 20 have a nearly null probability to occur. Reproducing such a behavior requires an adequate structure since a fully connected model would give a larger probability mass to short FPT. POMMStruct adequately models this shifted distribution, *i.e.* the induced structure supports well the dynamics of the target model. POMMStruct also correctly models respectively the three modes and the bell-shaped dynamics of target distributions 3 and 4 even if the empirical distributions are somewhat noisy. Target distributions 5 and 6 exhibit long-term dependencies in the target model as short passage times have a rather small probability while the major modes are located round passage time 60. In the last row of Figure 5.13, it can be observed that POMMStruct tends to exaggerate the early peak of the target distributions. This may result from a partial overfitting of the empirical distributions where early peaks are over-estimated. To sum up, POMMStruct is able to model a wide range of FPT dynamics including multimodal distributions, shifted distributions and long-term dependencies. The correct modeling of these distributions is achieved through the induction of an adequate model topology.

### 5.6.5 Influence of the parameters

In this section, the influence of the hyperparameters, *i.e.* the initial model order and the number of selected features, is studied. The number $n_r$ of restarts and the precision parameter $\epsilon$ can also be considered as hyperparameters. Using a larger number of restarts essentially helps to avoid the local minima of the FPT likelihood function. The precision parameter $\epsilon$ is used in the stopping criterion of POMMStruct to check if a significant improvement has been made. The influence of these parameters has not been systematically checked, however, we have observed that using $n_r \geq 3$ and $\epsilon \leq 10^{-6}$ provides generally good results. Therefore, all our experiments were carried out using $n_r = 3$ and $\epsilon = 10^{-6}$. The learning curves presented hereafter are produced by extracting samples of growing sizes from the training set. For each data size (except for 100%), 10 samples have been randomly extracted from the training set in order to produce averaged results with standard deviations. Performances, in terms of FPT divergence and of perplexity, are measured on a validation set obtained by extracting randomly 25% of the training data. When measuring perplexities, the probabilistic parameters of models learned by POMMStruct have been

reestimated with the Baum-Welch algorithm without adapting the model structure.

### Influence of feature selection

This section studies the influence of feature selection on (i) the FPT divergence, (ii) the perplexity, (iii) the size of the model and (iv) the CPU time. We have also evaluated the influence of feature weighting using the JS divergence (see section 5.5.1). It appeared that slightly better performances are obtained *without* feature weighting on the considered datasets. However, in a preliminary experiment with sequences generated from the POMM $H_\theta$ displayed in Figure 5.7 with $\theta = 0.95$, using feature weighting exactly matches the target structure whereas the standard approach returns a model with 8 states (instead of 7) which does not fit as well the target dynamics. The top and bottom parts of Figure 5.14 respectively show the influence of the number of selected features on the FPT divergence and on the perplexity. Results on validation data are shown when 10%, 20%, 50%, 70% and 100% of features[10] are used.

As expected, using more features decreases the FPT divergence (which is averaged over all features) but good results are already obtained when 50% of features are used. We can therefore conclude that fitting a subset of features is sufficient to fit the complete FPT dynamics of the target process. This also underlines the effectiveness of our feature selection strategy which is based on the JS divergence (see section 5.5.1). Besides, increasing the number of features also improves the perplexity. The top and bottom parts of Figure 5.15 respectively show the influence of feature selection on the size (in terms of number of transitions) of the obtained model and on the CPU time of POMMStruct. In general, increasing the number of features provides models with a larger number of transitions. For instance, when the full training set is used, fitting 10% of features results in a model with 90 transitions whereas fitting all the features provides a model with 131 transitions. The last plot shows the evolution of the CPU time in function of the proportion of selected features. CPU time increases linearly with the number of features. This is consistent with the computational bounds given for the POMMPHit algorithm[11] (see section 5.5.2).

[10] The total number of features is $p = |\Sigma|^2$.

[11] At each iteration of the POMMStruct algorithm, the computational complexity is dominated by the complexity of POMMPHit.

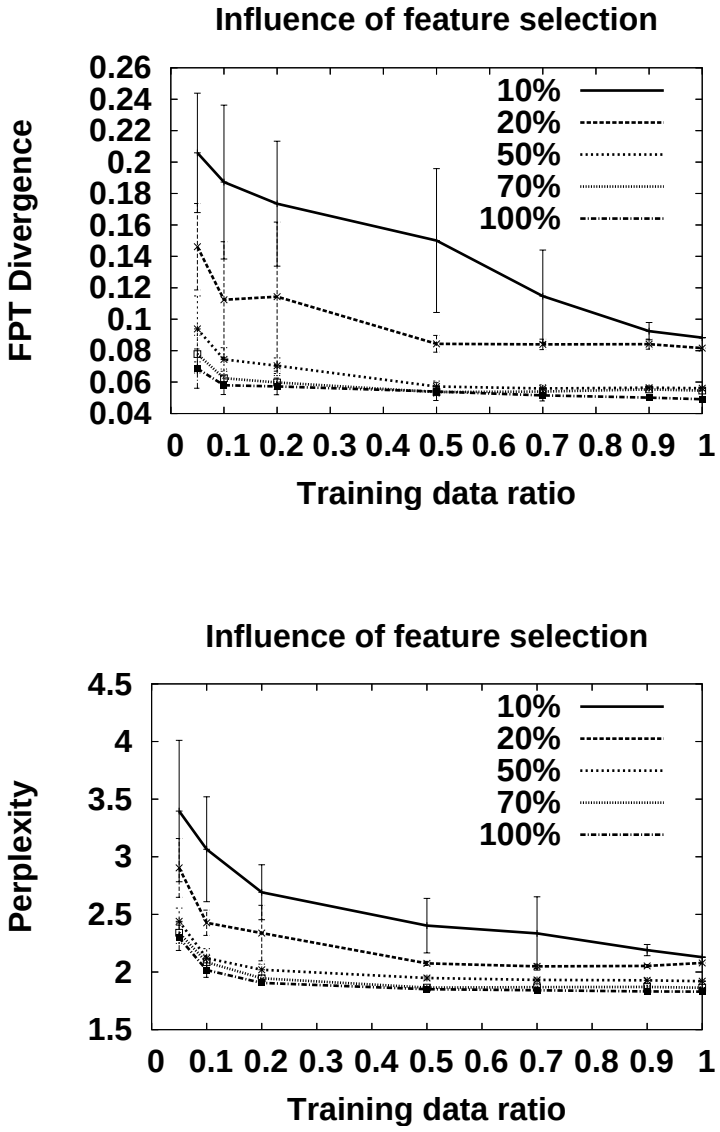



Figure 5.14: The influence of the number of features, *i.e.* FPT pairs and on the FPT divergence (top), the perplexity (bottom) evaluated using the `GenDep` validation data.

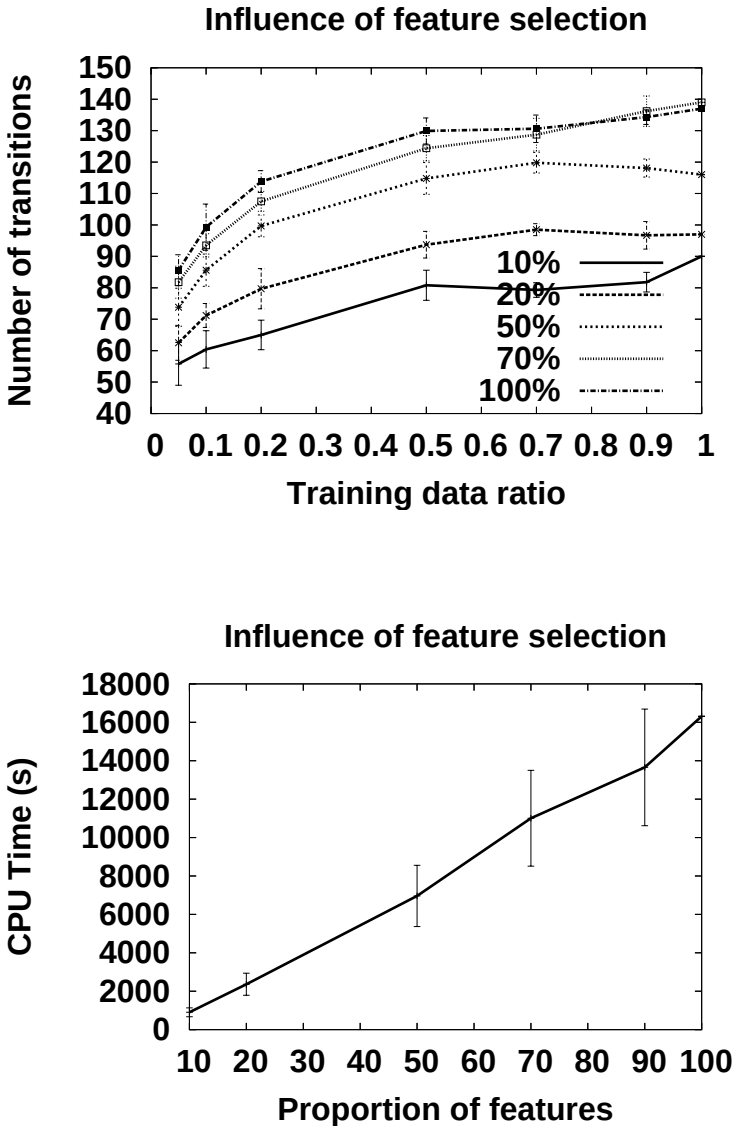



Figure 5.15: The influence of the number of features, *i.e.* FPT pairs, on the number of transitions (top) and on the run-time (bottom) evaluated using the `GenDep` training data.

**Influence of initial order**

This section investigates the influence of the order of the initial model on (i) the FPT divergence, (ii) the perplexity, (iii) the size of the model and (iv) the CPU time. For computational time reasons we have only initialized POMMStruct with order 1 and 2 MC (see section 5.5.1). For the `GenDep` and `CopChrom` datasets, using an order 2 MC provides slightly worse FPT divergences and perplexities than with an order 1 MC. For the `Splice` dataset, better results are obtained using an order 2 initial MC. The `Splice` dataset has a much simpler dynamics than the two other datasets: it consists of short passage times (the average FPT for each feature is less than 5) and only two features, (`a,t`) and (`t,g`), have a bimodal empirical FPT distribution. Therefore, we believe that using an order 2 initial MC improves the fit to these local dependencies. The top and bottom parts of Figure 5.16 respectively show the influence of the initial model order on the FPT divergence and on the perplexity measured on the `Splice` validation data.

When a few data is available, an initial order of 1 provides better results. Indeed, an initial order 2 MC has more parameters to estimate (see the top part of figure 5.17) than an order 1 MC and therefore reliable estimations require more data. When more than 20% of the training data are used, the (relatively small) benefits of the order 2 initial MC can be observed. The top and bottom parts of Figure 5.17 respectively show the influence of the initial order on the number of transitions and on the CPU time of POMMStruct. As expected, initializing POMMStruct with an order 2 MC provides a larger model than with an order 1 MC. Let us recall that using an order 2 initial MC initially multiplies the number of transitions by a factor $|\Sigma|^2$ with respect to an order 1 MC. The difference is most noticeable when a small amount of data is available. In this case, the estimated model is close to the initial model as a more complex model is likely to be rejected by validation data due to overfitting (see 5.5.1 for the model size selection). Besides, using a higher order initial model clearly increases the computational run-time. This follows from the larger number of parameters to estimate.

### 5.6.6 Comparative results

This section presents comparative results with the Baum-Welch algorithm (see section 4.1.1) and the Bayesian state merging algorithm due to Stolcke (see section 4.2.2). The hyperparameters of all the methods have been selected on a validation set obtained by holding out 25% of the training data. The Baum-Welch algorithm is applied on fully connected graphs of increasing sizes as detailed in section 4.1.1. For the `GenDep` dataset, the considered

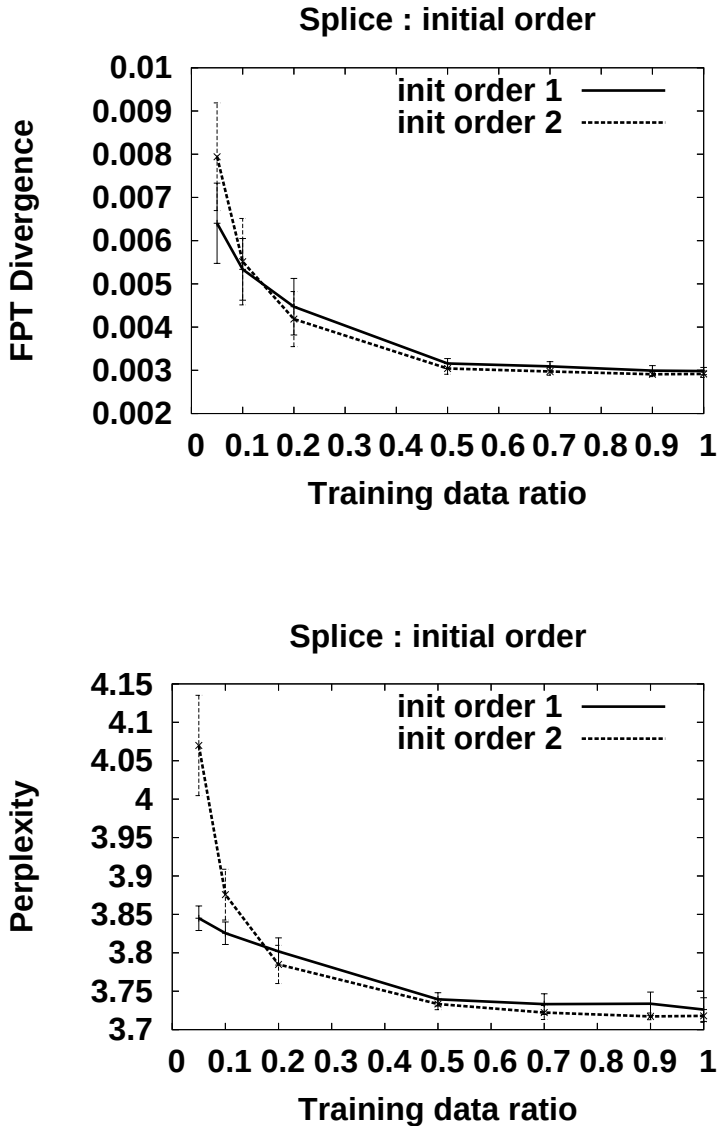



Figure 5.16: The influence of the initial model order on the FPT divergence (top) and on the perplexity (bottom) evaluated using the `Splice` validation data.

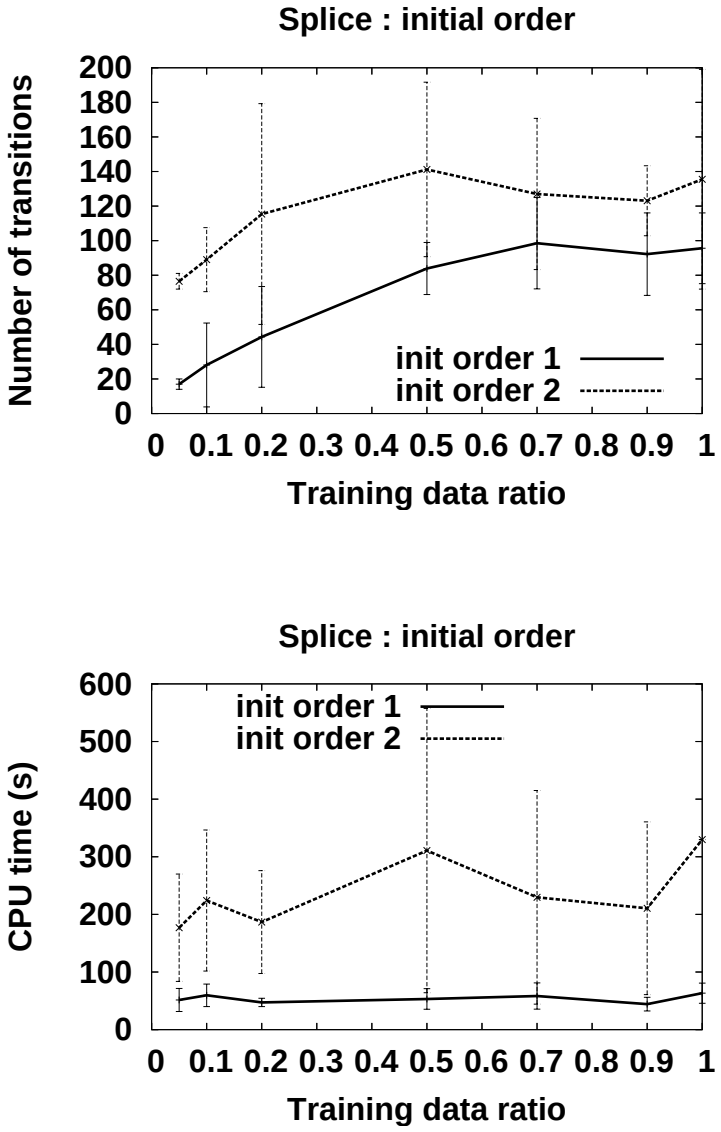



Figure 5.17: The influence of the initial model order on the number of transitions (top) and on the run-time (bottom) using the `Splice` training data.

graph sizes are $k/4, k/2, k$ and $3k/2$ where $k$ is the number of states in the target model. For the `Splice` and `CopChrom` datasets the following graph sizes are used: 2, 5, 10, 20 and 30. For each considered model size, three different random seeds are used and the model having the largest likelihood is kept. Additionally, a transition trimming procedure, based on the transition probabilities, has been used. The optimal model size is selected using the validation data. The Bayesian state merging technique of Stolcke has been reimplemented according to the setting described in section 3.6.1.6 of [121] (see also section 4.2.2). The *effective sample size* parameter, defining the weight of the prior versus the likelihood, has been tuned[12] in the set $\{1, 2, 5, 10, 20\}$ on validation data. Models estimated with this approach contain a final silent state to predict the end of sequences. Such a state is not present in the models resulting from POMMStruct or Baum-Welch. In order to get comparable measures, the final silent state is removed and the model is normalized so as to satisfy properness constraints. The POMMStruct algorithm is initialized with an order $r \in \{1, 2\}$ MC. All observed FPT pairs are considered (*i.e.* $p = |\Sigma|^2$) without feature weighting. Whenever applied, the POMMPHit algorithm (used in POMMStruct to estimate the model parameters) is initialized with three different random seeds and the parameters leading to the largest FPT likelihood are kept. The optimal model size and the initial model order are selected on the validation set. In addition to the stopping criterion presented in section 5.5.1, we have limited the maximum number of iterations (*i.e.* state additions) to 20. Performances are assessed using the FPT divergence and the perplexity (see section 5.6.2). When measuring perplexities, the probabilistic parameters of models learned by POMMStruct have been reestimated with the Baum-Welch algorithm without adapting the model structure.

Figure 5.18 shows learning curves on the `GenDep` test sequences which are drawn from an artificial target model with 32 states and 24 symbols. POMMStruct clearly outperforms its competitors in terms of FPT divergence. Furthermore, POMMStruct is able to correctly model the target dynamics even when a small amount of data is available. This has also been observed in the top part of Figure 5.14 when all the features are used. The algorithm of Stolcke performed rather well for small amounts of data but the performance does not improve much when more training data are available. The Baum-Welch technique poorly fits the FPT dynamics when a small amount data is used. However, when more data are available ($\geq 70\%$), it outperforms the approach of Stolcke. The bottom part of Figure 5.18 shows that POMMStruct has the best overall perplexity. When all the

[12] The fixed value of 50 recommended in [121] performed poorly in our experiments.

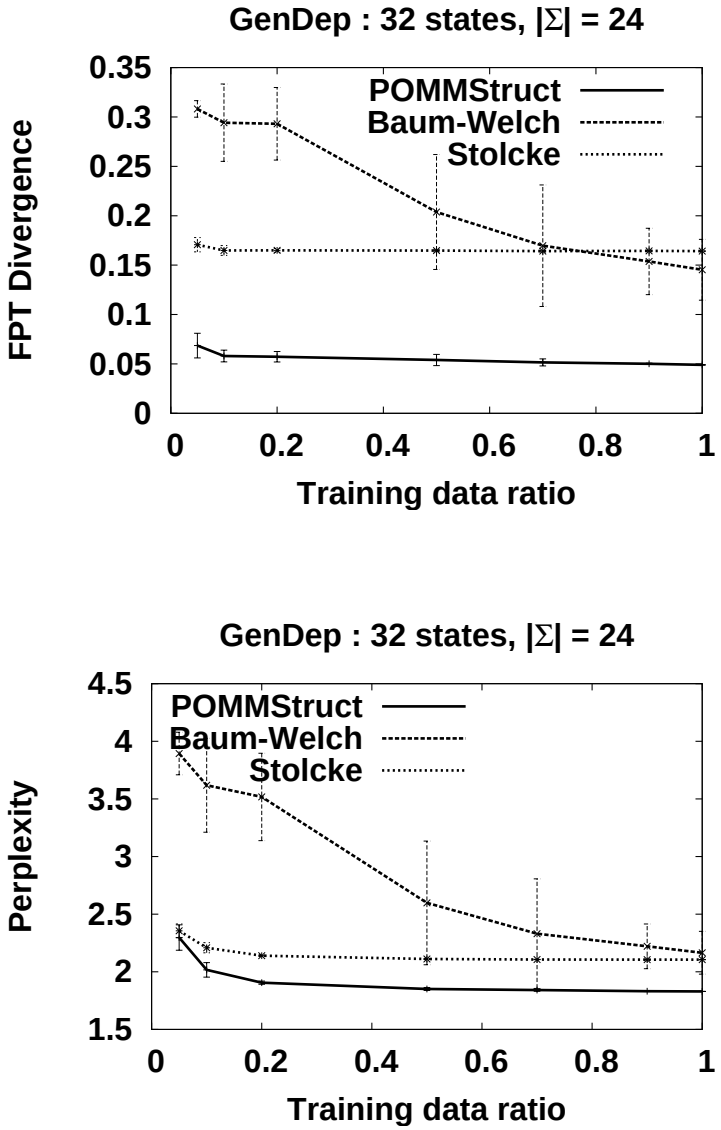



Figure 5.18: Comparative results using POMMStruct, Baum-Welch applied on fully connected graphs with transition trimming, and the Bayesian state splitting due to Stolcke on the `GenDep` test data. The train and test sequences have been been drawn from a target model defined on an alphabet of 24 symbols and containing 32 states.

training data are used, the approach of Stolcke obtains a slightly lower divergence than Baum-Welch. The relative perplexity increases with respect to the target model, used to generate the sequences, for POMMStruct, the approach of Stolcke and Baum-Welch are respectively 2%, 18% and 21%. When all the training data are used, the computational run-times are the following: 3.45 hours for POMMStruct, 2 hours for Baum-Welch and 35 minutes for Stolcke's approach. It should be noted that when 50% of the features are used, the run-time of POMMStruct is less than two hours while it still outperforms the competing approaches with respect to both criteria.

Figure 5.19 present results obtained on the `Splice` dataset. The FPT dynamics in these sequences is less complex than in other datasets, leading to smaller absolute JS divergences for all techniques. POMMStruct exhibits the best overall performance in terms of FPT divergence. Results on validation data have shown that better results are obtained when POMMStruct is initialized with an order 2 MC (see section 5.6.5). However, POMMStruct also outperforms its competitors when initialized with an order 1 MC. When more than 50% of the training data are used, the Baum-Welch algorithm performs slightly better than the technique of Stolcke. The perplexity obtained with POMMStruct and Baum-Welch are comparable (the relative perplexity increase of POMMStruct with respect to Baum-Welch is less than 1%) while the approach of Stolcke performs slightly worse (4% of relative perplexity increase). When all the training data are used, the computational run-times are the following: 25 minutes for Baum-Welch, 17 minutes for Stolcke's approach and 6 minutes for POMMStruct. In this case, POMMStruct runs faster than its competitors. This can be explained by the small size of the alphabet ($|\Sigma| = 4$) of the `Splice` sequences. Indeed, the computational complexity of POMMStruct increases quadratically with the alphabet size when all features are used. Therefore, POMMStruct runs much faster on sequences defined on small alphabets.

Figure 5.20 display performances obtained on the `CopChrom` dataset. The FPT dynamics in these sequences is rather complex with many peaks which makes the task rather challenging. Results have been computed using a standard tenfold cross-validation for each one of the 22 chromosome types. POMMStruct obtains the lowest FPT divergence on each of the 22 categories. The relative divergence increase, averaged on the 22 categories, with respect to POMMStruct is 51% for Baum-Welch and 64% for the approach of Stolcke. Concerning the perplexity, Baum-Welch has the best overall performance but POMMStruct has the lowest perplexity on the chromosome types 2, 6, 12, 17 and 20. The perplexity increase, averaged on

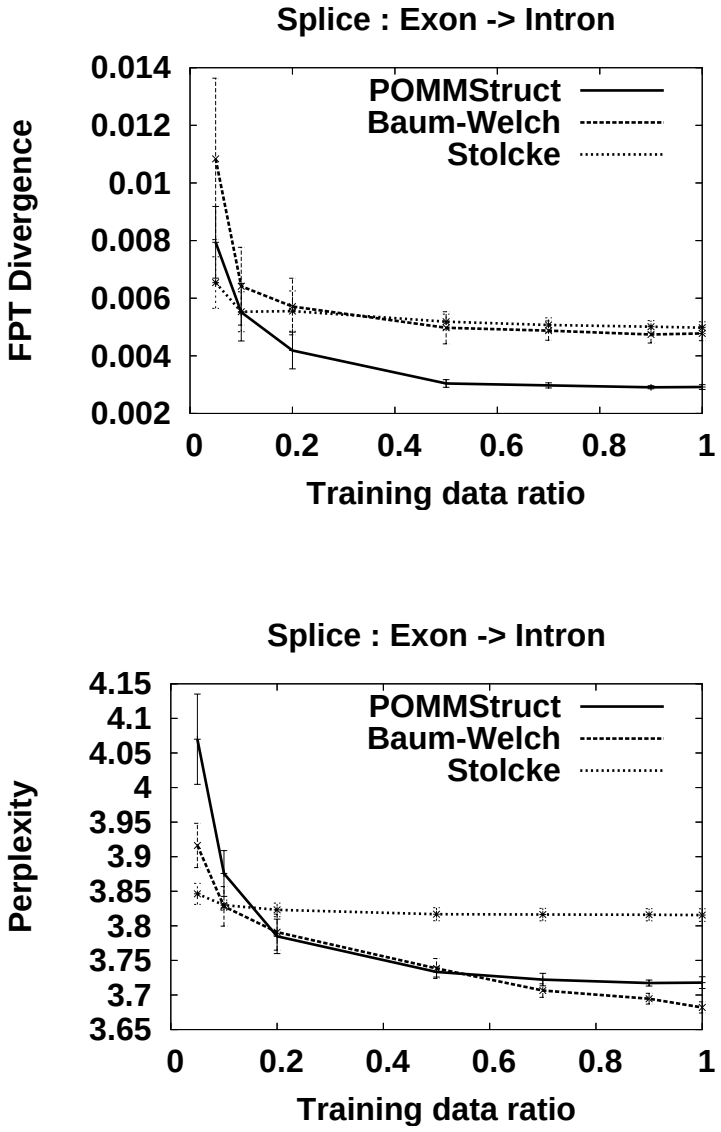



Figure 5.19: Comparative results using POMMStruct, Baum-Welch applied on fully connected graphs with transition trimming, and the Bayesian state splitting due to Stolcke on the `Splice` test data.

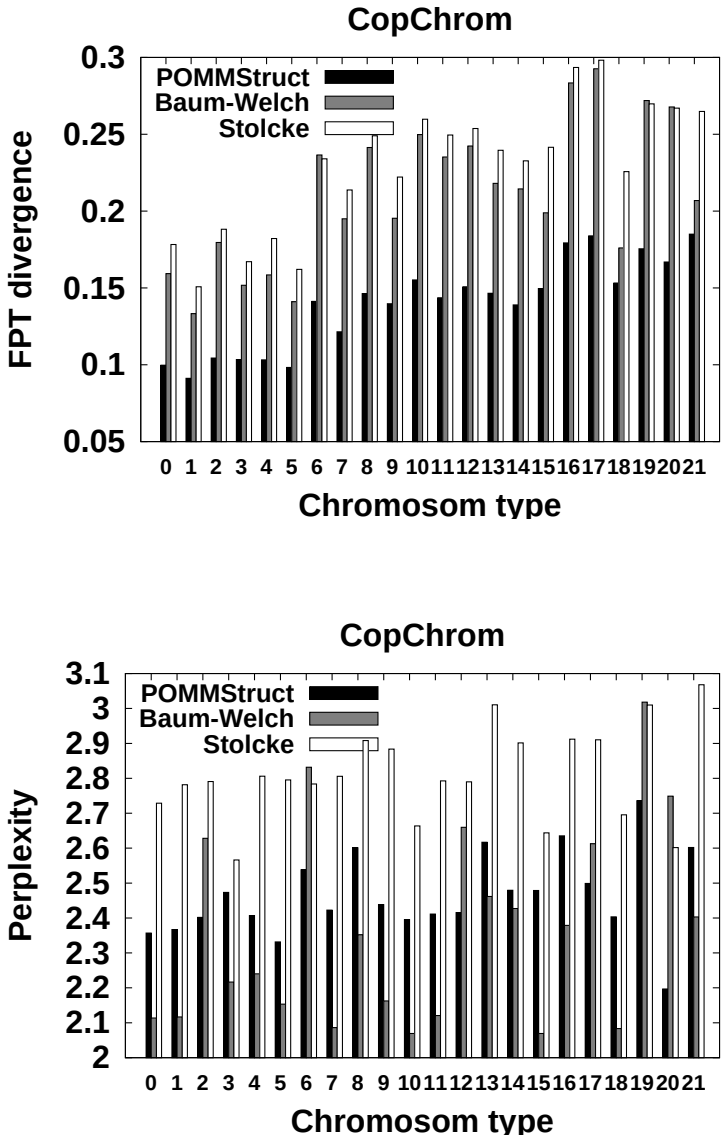



Figure 5.20: Comparative results using POMMStruct, Baum-Welch applied on fully connected graphs with transition trimming, and the Bayesian state splitting due to Stolcke on the `CopChrom` data. Performances are evaluated separately for each of the 22 chromosome types (or categories) using a ten-fold cross-validation.

all the categories, with respect to Baum-Welch is 4% for POMMStruct and 19% for the Stolcke's approach. The computational run-times, averaged on all the categories, are the following: POMMStruct and Stolcke's approach both take 8 minutes whereas Baum-Welch takes 23 minutes.

To sum up, the comparative results presented above show the superiority of POMMStruct to model the FPT dynamics observed in sequences. This is mostly noticeable on the `GenDep` and `CopChrom` datasets where the dynamics is more complex than in the `Splice` dataset. The models obtained with POMMStruct and reestimated using Baum-Welch have a lower perplexity than those estimated with the approach of Stolcke. POMMStruct obtains a lower perplexity than Baum-Welch on the `GenDep` dataset whereas both approaches have very close perplexities when applied to the `Splice` and `CopChrom` datasets.

### 5.6.7 Implementation issues

A prototype of the proposed induction algorithm, POMMStruct (see section 5.5.1), has been implemented in the Python 2.4 language whereas the inner PH fitting procedure, POMMPhit (see section 5.5.2), has been developed in the ANSI C language. In the one hand, the Python programming language offers convenient string manipulation procedures, useful to extract FPT in the sequences. Moreover, POMMStruct also takes benefit from the object-oriented layer of Python to define objects like POMMs or other intermediate structures. In the other hand, the major computational burden takes place in the POMMPhit estimation algorithm which is called each time a new state is added to the model. For this reason, we have chosen to develop an external program written in a very efficient second generation programming language, namely the ANSI C language. Presently, both programs communicates via Unix pipes and files. A more elegant and efficient bridge would rely on the SWIG development tool[13] which connects C or C++ programs to a variety of high-level programming languages such as Python. The Baum-Welch algorithm with transition trimming (see section 4.1.1) used in our comparative experiments follows the same design as POMMStruct: the inner parameter estimation routine has been coded in ANSI C whereas the outer structural procedure including transition trimming has been developed in Python. The Bayesian state merging algorithm due to Stolcke (see section 4.2.2) has been fully programmed in ANSI C. In this context, the beta function is computed using a routine proposed in the "Numerical Recipes in C" [104]. The implementations of POMMStruct as well as

[13]This tool can be freely obtained at `http://www.swig.org/`.

the Baum-Welch algorithm and Stolcke's method are available on the UCL Machine Learning Group (MLG) website: `http://www.ucl.ac.be/mlg/`.

## 5.7 Summary and discussion

This chapter has presented a novel approach to the structural induction of HMMs. The proposed technique, POMMStruct, builds a model reproducing the First Passage Times (FPT) dynamics observed in a sample drawn from some unknown target process. Unlike $N$-grams (*i.e.* contiguous sequences of length $N$), FPT features are not local as there is no fixed maximum time (*i.e.* number of steps) between two events. Furthermore, the FPT distributions contain relevant informations, such as the presence of dominant path lengths or long-term dependencies, about the structure to be learned.

We have introduced a new kind of model called Partially Observable Markov Models (POMMs). These models are HMMs constrained such that each state can only emit a single symbol with probability one. The state set of POMMs can be partitioned such that the states emitting the same symbol are gathered in blocks. Importantly, we have shown that POMMs and HMMs generate the same class of distributions on strings. Our induction algorithm POMMStruct uses POMMs as target representation since FPT are conveniently computed in these models while the expressiveness of HMMs is preserved. Links between POMMs and standard Markov chains (MC) have also been investigated. We have shown that POMMs are equivalent to lumped MC, *i.e.* MC with a partitioned state set and such that only the block identities are observed. Sufficient and necessary conditions for a POMM to define a MC were also given via the notion of *lumpability*.

The distributions of FPT in POMMs (or equivalently in HMMs) have been studied. We have shown that these distributions are of phase type (PH). Moreover, we have motivated the use of the FPT in model induction by showing that they are informative about the structure to be learned. For instance, multi-modal FPT distributions reveals the presence of dominant path lengths between blocks in the model. Importantly, these structural informations are directly available in the learning sample. In a nutshell, accurately modeling the FPT dynamics requires an adequate topology. Such a topology is learned incrementally by our induction algorithm which is driven by the fit to the observed FPT.

We have then focused on modeling sequential processes including long-term dependencies, which are processes for which an outcome can depend on the realization of some event at a much earlier time. We have shown that finite order Markov chains (MC) poorly model such dependencies, especially when the conditioning event can be arbitrarily far in the past. More powerful models such as HMMs or POMMs are required to model such dependencies. Nevertheless, we have shown that using an adequate structure is crucial for reproducing such dependencies. This follows from the Perron-Frobenius theorem (see section 1.2) which implies that a model can only have a long-term memory if the second largest eigenvalue of its transition matrix is large. In general, fully connected graphs with uniformly randomized transitions probabilities have a very low second largest eigenvalue. This theoretically explains why the Baum-Welch algorithm applied to fully connected graphs (see section 4.1.1) fails to model long-term dependencies. Links between long-term dependencies and FPT have been highlighted. We have shown that the FPT directly measure the time between conditioning and conditioned events. Therefore, long-term dependencies are reflected in the FPT dynamics of a process. This makes the FPT a suitable criterion to fit these dependencies.

The POMMStruct algorithm for POMM induction has been introduced. This technique learns the structure of POMMs and estimates jointly their parameters to fit the observed FPT. A new likelihood function relying on PH distributions is introduced. Unlike the traditional likelihood function, the FPT likelihood is concerned with times between events (*i.e.* emission of symbols) rather than with the complete generative process. POMMStruct is initialized with a standard MC, the order of which is a user parameter typically set to one. By default, POMMStruct fits the FPT associated to all possible pairs of symbols (`a,b`) where $\mathtt{a}, \mathtt{b} \in \Sigma$. Alternatively, a feature selection procedure is proposed to fit only a limited number pairs. Feature selection finds, via the Jensen-Shannon divergence, the pairs poorly fitted by the initial MC. At each iteration, a new state is added to the model and the additional parameters are estimated using a novel algorithm called POMMPHit. This is a maximum likelihood estimation based on EM and concerned with the FPT likelihood function. An interesting byproduct of POMMPHit is the expected number of times transitions are triggered while observing the sample FPT. This defines a very natural criterion to trim less frequently used transitions from the model. The optimal model size is selected on an independent set of validation sequences.

Finally, we have presented experimental results obtained with POMMStruct on several sequence modeling tasks. These experiments aimed at

(i) assessing the quality of the model structure learned with POMMStruct (iii) evaluating the ability of POMMStruct to model complex FPT dynamics, (iv) check the influence of hyperparameters and (v) compare the performances of POMMStruct with respect to Baum-Welch applied on fully connected graphs with transition trimming (see section 4.1.1) and the Bayesian merging approach due to Stolcke (see section 4.2.2). Firstly, experiments with data drawn from known artificial target models have shown that POMMStruct provide models close to the targets in terms of number of states per block but with slightly more transitions. Then, we have shown that POMMStruct is able to model a wide range of FPT dynamics including multi-modal distributions, shifted distributions and long-term dependencies. The influence of feature selection and of the initial model order has been studied. It was shown that using a limited amount of features can model well the complete FPT dynamics observed in the sample while significantly reducing the computational run-time. Using a higher order initial MC does not systematically yield better results but it can improve the fit to local dependencies present, for instance, in DNA sequences. Finally, comparative results have shown that POMMStruct is better suited to fit a process with a complex FPT dynamics than Baum-Welch or Stolcke's approach. Models obtained with POMMStruct and reestimated with Baum-Welch have a lower perplexity than those estimated with the approach of Stolcke on all the considered datasets. POMMStruct does not always offer a lower perplexity than Baum-Welch, however the FPT dynamics observed in the sample is more accurately modeled by our technique.

We now discuss the limitations of our approach and possible future researches in that perspective. A first remark concerns the use of POMMs instead of HMMs in our induction technique. While being equivalent to HMMs, POMMs are often less concise than HMMs and therefore harder to interpret. We have provided a technique for converting a given HMM into a POMM but the converse transformation has not been investigated. When converting a HMM into an equivalent POMM, the resulting model has, by construction, additional constraints linking its parameters (see theorem 8). A POMM satisfying such constraints can readily be transformed into a more concise HMM. POMMStruct could be adapted to satisfy these constraint and to return a HMM rather than a POMM. Alternatively, one could adapt our reestimation procedure, POMMPHit, to apply to HMMs. Finally, one could use the technique proposed in [7] to get a canonical representation of the estimated POMM. This technique would however return a generalized HMM, *i.e.* a relaxed HMM with parameters in $\mathbb{R}$, rather than a standard HMM.

Up to now, POMMStruct fits the observed FPT dynamics between individual symbols. An interesting extension would consider higher order events, that is modeling the FPT between occurrences of subsequences in the input sequences. In the context of sequence classification (see chapter 7), such extended FPT are modeled using PH distributions. The major difference with the current approach is that each subsequence pair is fitted with a distinct model. These extended features are shown to be discriminant in sequence classification tasks. We believe that POMMStruct could also take benefit from these features. For instance, it could be used to model long-term dependencies between aggregated events (*i.e.* subsequences).

Besides, POMMPHit makes an independence assumption concerning the symbol pairs. More precisely, it is assumed that the FPT for a given pair are independent from the FPT associated to the other pairs. Taking the possible FPT dependencies into account could also be investigated in a future work.

Finally, smoothing issues are an important problem for HMM or POMM induction. That is, the learning sample is often very sparse and consequently, numerous possible events are given a null probability by the estimated models. This can cause problems while computing the perplexity of sequences containing such events as the log-likelihood becomes unbounded. Smoothing techniques aim at spreading the probability mass to unobserved or rare events. Approaches towards this objective include back-off smoothing (see section 1.5) applied to Probabilistic Automata (PA) [87], symbol clustering [44] and error-correcting techniques [42]. All these techniques are concerned with smoothing the sequence likelihood. Alternatively, one could focus on smoothing the FPT. That is, one wants to spread the probability mass to rare or unobserved FPT. In section 7.3.2 we will show that modeling the FPT with PH distributions can be thought of as a smoothing technique where the amount of smoothing is controlled by the number of phases. The number of phases corresponds here to the number of states in the POMM. One could therefore control the number of state additions towards this smoothing objective.

# Part III

# Sequence classification

# Chapter 6

# An overview of sequence classification methods

*This chapter reviews some important methods for solving discrete sequence classification problems. Given a training set of $n$ examples $Tr = (S, Y) = \{(s^1, y_1), \ldots, (s^n, y_n)\}$ where $s^i$ is a sequence defined on an alphabet $\Sigma$ and $y_i \in \mathcal{Y}$ is its label (or its class), one wants to learn a function $f : \Sigma^* \rightarrow \mathcal{Y}$ for predicting the label of new sequences. Practical applications of this task range from the recognition of splicing sites in DNA sequences to musical pieces classification. We distinguish here two general kinds of approaches to this problem: (i) generative techniques where the classification relies on a probabilistic model of the data and (ii) discriminative methods directly focusing on the decision function.*

*Generative techniques aim at modeling the generative process of the data $P[S, Y]$. A new sequence $s$ can be classified using a maximum likelihood or a maximum a posteriori (MAP) decision rule. The Naive Bayes classifier is perhaps the most simple and popular generative technique for classifying sequences. In such a classifier, symbols are assumed to be generated independently. The locations or co-locations of symbols in the sequences are not taken into account in such a classifier. Consequently, a random permutation of the symbols in sequences would have no effect on the resulting model as the sufficient statistics, i.e. the symbol frequencies, remain unchanged after such an operation. $N$-grams, or equivalently order $N-1$ Markov chains (MC), can be considered as an extension of Naive Bayes, modeling the dependencies between successive events in a window of length $N$. Estimation techniques for such models have been presented in section 1.5. HMMs, presented in chapter 2 are a generalization of MC modeling even more general dependencies in sequences. In section 7.4.1, we propose a novel generative classification technique based on phase-type (PH) distributions modeling the first passage times (FPT) in sequences. In section 5.3, it has been shown*

*that the FPT between symbols in HMMs define PH distributions. Hence, using these distributions for characterizing sequences allows us to include a part of the powerful expressiveness of HMMs in our classification techniques.*

*In contrast with generative classification techniques, discriminative approaches directly focus on the decision function used to classify sequences. To do so, models such as MC and HMMs can be trained so as to maximize the conditional likelihood (see section 4.3.1 for HMMs) rather than the data likelihood. Besides, non-probabilistic methods such as multiple layer perceptrons, decision trees, nearest neighbors and kernel methods can also be applied to classify sequences. Kernel methods (see chapter 3) are now considered as the state-of-the-art in sequence classification. They basically consist of a Support Vector Machine (SVM) classifier along with a specific kernel computing the similarity between sequences. String or sequence kernels embed sequences into some relevant feature space based on characteristics such as counts of subsequences. Finally, one can couple generative and discriminative approaches by using a kernel relying on a probabilistic model to compute similarities. In section 7.4.2, we propose a novel string kernel relying on a probabilistic model of the FPT, namely the PH distributions.*

*This chapter is organized as follows. Section 6.1 reviews some generative classification techniques including Naive Bayes and $N$-gram classifiers (subsection 6.1.1) as well as a technique based on the Lempel-Ziv78 (*LZ78*) algorithm used in data compression (subsection 6.1.2). Section 6.2 reviews discriminatively trained MC (subsection 6.2.1) as well as state-of-the-art string kernels (subsection 6.2.2).*

## 6.1 Generative approaches

This section is concerned with generative methods for classifying discrete sequences. Such techniques build a probabilistic model of the data and predict the label of new sequences according to a maximum likelihood or a Maximum *a posteriori* (MAP) decision rule. In this context, one of the simplest methods is the Naive Bayes classifier [68] which is equivalent to a unigram model. Sequence classification with Naive Bayes only relies on the frequencies of individual symbols within the different classes. Such a technique was shown to provide good results in text classification problems. $N$-gram extends Naive Bayes by modeling dependencies between successive events in a sliding window of length $N$. The estimation of such models is not straightforward as the number of distinct $N$-grams grows exponentially fast with $N$ and a huge amount of data is required to get reliable estimates. To face this issue, one can use robust estimation techniques presented in section

1.5. Such estimation techniques define variable order MC by using a back-off smoothing scheme. Alternatively, lossless compression algorithms such as the classical Lempel-Ziv-78 (LZ78) can be used in sequence prediction and classification problems. These techniques also define variable order MC and were shown to produce good results in sequence classification tasks [9]. HMMs introduced in chapter 2 are more powerful than finite order MC as they can model more general dependencies, such as long-term dependencies (see section 5.4), in sequences. However, such models are more difficult to learn than $N$-grams and do not systematically yield better classification results. In sections 7.4.1 and 7.4.2 we show how PH distributions can be efficiently used to perform sequence classification. In section 7, we show that these novel methods compare favorably with state-of-the-art classification techniques.

Subsection 6.1.1 briefly reviews the Naive Bayes and $N$-gram classifiers while subsection 6.1.2 presents a classification technique based on the LZ78 algorithm.

### 6.1.1 Naive Bayes and N-grams classifiers

According to the Bayesian decision theory, the highest expected classification accuracy, or equivalently the minimum expected error rate (also called the functional risk in section 3.1.1), is achieved by predicting the class $y \in \mathcal{Y}$ having the highest posterior probability $P[y \mid s]$. The class posteriors can be computed using the Bayes theorem, which yields the following decision rule to classify a new sequence $s$:

$$y^* = \arg\max_{y \in \mathcal{Y}} P[y \mid s] \text{ where } P[y \mid s] = \frac{P[s \mid y]P[y]}{\sum_{y \in \mathcal{Y}} P[s \mid y]P[y]} \tag{6.1}$$

Since the denominator is the same for all the classes, it can be ignored from the maximization. The decision rule displayed in equation (6.1) is guaranteed to yield the lowest expected error rate when the distributions $P[S \mid Y]$ and $P[Y]$ are known. In practice, these distributions have to be estimated from data. The prior probability of a class $y \in \mathcal{Y}$ is simply estimated as the relative proportion of sequences labeled with $y$ in $Tr$:

$$P[y] = \frac{|\{(s^i, y_i) \in Tr \mid y_i = y\}|}{n} \tag{6.2}$$

The estimation of $P[S \mid Y]$ is somewhat more challenging as there are exponentially many[1] distinct sequences for a given length. Consequently, obtaining a reliable estimate of the sequence distribution using simple count

[1] The number of sequences of length $l$ defined on an alphabet $\Sigma$ is $|\Sigma|^l$.

ratios, as in equation (6.2), is practically infeasible as it would require a huge amount of training data. In order to estimate the distribution $P[S \mid Y]$, the naive Bayes classifiers makes the assumption that the symbols occur independently in sequences. Therefore, the probability, or the likelihood, of a sequence $s = s_0 \dots s_l$ given a class $y$ is computed as follows

$$P[s \mid y] = \prod_{i=0}^{l} P[s_i \mid y] = \prod_{\mathtt{a} \in \Sigma} P[\mathtt{a} \mid y]^{C_s(\mathtt{a})} \tag{6.3}$$

where $C_s(\mathtt{a})$ counts the number of occurrences of $\mathtt{a}$ in the sequence $s$. Consequently, the estimation of $P[s \mid y]$ is reduced to estimating each $P[a \mid y]$ independently. The Laplace estimate for such probabilities is given hereafter.

$$P[\mathtt{a} \mid y] = \frac{1 + C^y(\mathtt{a})}{|\Sigma| + \sum_{\mathtt{a} \in \Sigma} C^y(\mathtt{a})} \tag{6.4}$$

where $C^y(\mathtt{a})$ counts the number of occurrences of $\mathtt{a}$ in the sequences labeled with $y$. A virtual count is added to the numerator in order to insure a strictly positive probability for symbols not observed in the training sequences labeled with $y$. Indeed, if a test sequence contains a symbol $\mathtt{c}$ unobserved in the training sequences labeled with $y$, the probability $P[\mathtt{c} \mid y] = 0$ would be propagated through the probability products in the right part of equation (6.3), eventually defining $P[s \mid y] = 0$. The term $|\Sigma|$ is added to the denominator in order to define a proper conditional distribution over $\Sigma$.

The naive Bayes classifier is easily extended by estimating the distribution $P[S \mid Y]$ using $N$-grams rather than frequencies of individual symbols corresponding to unigrams. This finer modeling allows one to take dependencies within a finite range into account. Probability smoothing is even more crucial here as there are numerous unobserved $N$-grams in the training data. The KN back-off smoothing presented in section 1.5 can be applied to get (more) reliable estimates of the $N$-grams probabilities. The obtained model actually defines a variable order MC, the order of which ranges from 0 to $N - 1$. These models have been assessed in sequence classification problems and are shown to be very robust to noisy training data [41]. The core of the computational load of smoothed $N$-gram classifiers consists of counting all contiguous subsequences up to length $N$ in the training set $Tr$. Consequently, $N$-gram classifiers have a time complexity $\mathcal{O}(N \, ||S||)$ where $||S||$ is the cumulated length of all sequences in the training set.

### 6.1.2 A compression-based approach

There is a close relation between lossless compression and sequence prediction algorithms as, according to the Shannon theory, data can be encoded

with a smaller number of bits if the generative distribution (or a model of this distribution) is available. Furthermore, it turns out that any lossless compression algorithm can be used for prediction and conversely. This section presents a technique based on the LZ78 compression algorithm [142] used in a variety of compression and archiving utilities. Compression is achieved by replacing portions of data with references to previously encountered fragments stored in a lexicon. This algorithm defines a mediocre predictor whenever applied to new sequences but it turns out that this technique along with refinements proposed in [97] performs very well on sequence classification problems such as protein categorization [9].

The LZ78 algorithm builds a variable order MC from a data sample $S$ by performing the following steps. First, the sequences $s \in S$ are parsed as non-overlapping fragments in an incremental fashion. Note that considering only non-overlapping fragment yields in general a crude MC estimation, nevertheless, this parsing is well-suited for compression purpose. For the sake of simplicity, we explain this parsing technique through an example consisting of the sequence $s = \texttt{banana}$. A lexicon $\mathcal{L}$ is initialized with the empty sequence $\varepsilon$. The sequence $s$ is then scanned from left to right and the parser adds successively the smallest subsequences not yet present in $\mathcal{L}$. For instance, when the parser reaches the first `n` in $s$, the lexicon is $\mathcal{L} = \{\varepsilon, \texttt{b}, \texttt{a}, \texttt{n}\}$. When reaching the second `a`, the parser adds `an` in $\mathcal{L}$ as `a` was already present in the lexicon. The parsing continues after the second `n` as no overlapping subsequences are considered. The remaining of the sequence contains no other subsequence to add in the lexicon, which is finally $\mathcal{L} = \{\varepsilon, \texttt{b}, \texttt{a}, \texttt{n}, \texttt{an}\}$. The same procedure applied to $s = \texttt{abcbcd}$ would return $\mathcal{L} = \{\varepsilon, \texttt{a}, \texttt{b}, \texttt{c}, \texttt{bc}, \texttt{d}\}$. The lexicon is efficiently represented using a prefix tree built as follows. Initially, the tree consists of a root node labeled with $\varepsilon$ and having $|\Sigma|$ child leaves. Each node of the tree maintains a counter which is always equal to 1 for the leaves and equal to the sum of the children's counters for the internal nodes. Given a newly parsed fragment $s'$, the tree is (deterministically) traversed from the root until reaching a leaf $q$. This leaf is then transformed into an internal node with $|\Sigma|$ child leaves and the counters are updated backwards along the path from $q$ to the root node. Such a prefix tree constructed from our example sequence $s = \texttt{banana}$ is depicted in Figure 6.1. The prefix tree resulting from parsing the sample $S$ is used to compute conditional probabilities $P[\texttt{a} \mid h]$ as follows. The prefix tree is traversed from the root according to the history $h$. If a leaf is reached before $h$ is consumed, the traversal is continued from the root node according to the remaining symbols. If $q_h$ denotes the node (which can be an internal node or a leaf) reached at the end of the traversal and $q_{\texttt{a}}$ denotes the child of $q_h$ corresponding to symbol `a`, the conditional probability is computed as $P[\texttt{a} \mid h] = C(q_{\texttt{a}})/C(q_h)$ where $C(q)$ is the counter of

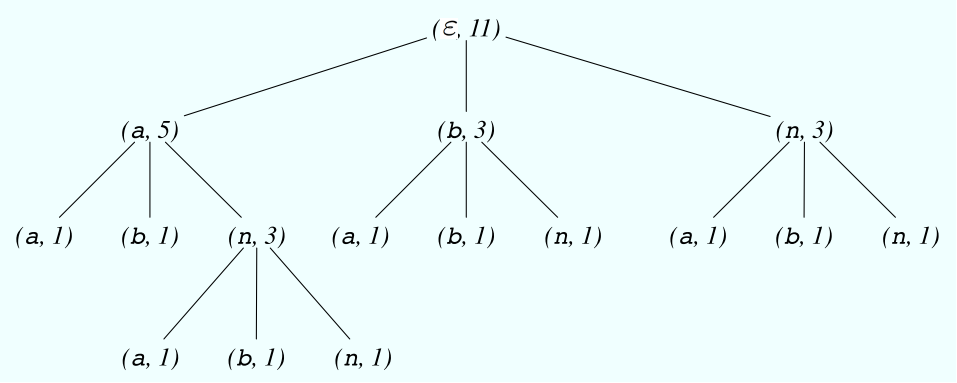



Figure 6.1: The tree constructed by LZ78 when applied to the sequence $s = \texttt{banana}$. LZ78 extracts the set of non-overlapping fragments of increasing sizes $\{\texttt{b}, \texttt{a}, \texttt{n}, \texttt{an}\}$ and constructs the prefix tree accordingly. Nodes are labeled with a pair $(\texttt{a}, i)$ where $\texttt{a}$ is a symbol and $i$ is the node counter which is equal to the sum of the children's counters for internal nodes and equal to one for the leaves.

node $q$. For instance, for computing $P[\texttt{b} \mid \texttt{an}]$ the tree is traversed as follows $\varepsilon \rightarrow \texttt{a} \rightarrow \texttt{n} \rightarrow \texttt{b}$ and the desired probability is $1/3$. To compute $P[\texttt{b} \mid \texttt{ab}]$, the traversal is $\varepsilon \rightarrow \texttt{a} \rightarrow \texttt{b} \rightarrow \varepsilon \rightarrow \texttt{b}$ and $P[\texttt{b} \mid \texttt{ab}] = P[\texttt{b} \mid \varepsilon] = 3/11$. This model clearly defines a variable order MC as the prediction of the next symbol may depend on a variable length history. Let us recall that this variable order MC is crudely estimated since only non-overlapping fragment have been considered and the observed frequency of these fragment are not taken into account.

Estimating a variable order MC with LZ78 essentially amounts to enumerating the non-overlapping fragments, what can be done in a time complexity $\mathcal{O}(nL)$ where $n$ is the number of sequences in the training set and $L$ is the length of the longest sequence in the training set. This low computational requirement is achieved at the cost of possibly unreliable probability estimations. On the one hand, not all the subsequences present in the input sequences are taken into account. In our example $s = \texttt{banana}$, the subsequence $\texttt{na}$ is not included in the lexicon nor in the prefix tree. This subsequence would however be very useful to compute the conditional probability $P[\texttt{n} \mid \texttt{na}]$. On the other hand, a loss of context can happen when a traversal is reset to the root of the tree. For instance, when computing $P[\texttt{b} \mid \texttt{ab}]$ one could rely on the more informed probability $P[\texttt{b} \mid \texttt{a}]$ rather than defining $P[\texttt{b} \mid \texttt{ab}] = P[\texttt{b} \mid \varepsilon]$. To face these issues, the authors of [97] have proposed to use techniques called *input shifting* and *back-shift*

*parsing*. Input shifting considers additional fragments by parsing the input sequences several times starting from different offsets. Besides, back-shift parsing attempts to improve the computation of conditional probabilities by guaranteeing a minimal context length. This improved LZ78 predictor has been shown to outperform other variable MC approaches when applied to protein classification tasks [9].

## 6.2 Discriminative approaches

This section presents discriminative techniques to the discrete sequence classification problem. In contrast with generative approaches, these techniques do not focus on how the data were generated but on a decision function allowing one to predict the label of sequences. In a probabilistic setting, the conditional distribution $P[Y \mid S]$ is estimated without resorting to a maximum likelihood or a MAP estimation of the joint data distribution $P[S, Y]$. In this context, we present a technique for estimating $N$-gram models which optimize the conditional likelihood. Moreover, kernel methods have been successfully applied to sequence classification problems by defining kernels embedding the sequences into some relevant feature space. In this context, we present two string kernels which are considered as the state-of-the-art in sequence classification. Finally, it should be noted that both generative and discriminative approaches can be unified using kernels which compute similarities based on generative models. This is the objective of the so-called *marginalization* and *Fisher* kernels [118]. In section 7.4.2, a novel marginalization kernel based on phase-type distributions is proposed to perform sequence classification.

Subsection 6.2.1 presents a method for training MC discriminatively while subsection 6.2.2 reviews kernel methods for sequence classification.

### 6.2.1 Discriminative Markov models

This section presents the discriminative counterpart of the generative $N$-gram classification approach described in section 6.1.1. This technique was proposed by Yakhnenko et al. [137]. The basic idea amounts to estimate the probabilistic parameters of $N$-gram models, or equivalently of order $N-1$ MC, so as to maximize the conditional likelihood of the data. Given a sequence $s = s_0 \dots s_l$ defined on an alphabet $\Sigma$, the posterior probability of a class $y \in \mathcal{Y}$ is given by

$$P[y \mid s] = \frac{P[s, y]}{\sum_{y' \in \mathcal{Y}} P[s, y']} \tag{6.5}$$

According to the Markov property, the joint probability $P[s, y]$ is provided by

$$P[s, y] = P[s_{0:N-2}, y] \prod_{k=N-1}^{l} P[s_k \mid s_{k-N+1:k-1}, y] \tag{6.6}$$

where $s_{i:j}$ denotes the subsequence of $s$ starting at index $i$ and ending at index $j$ and $l$ is the last index of the sequence $s$. Note that the factor $P[s_{0:N-2}, y]$ in equation (6.6) is a generalization of the initial probability $\iota_q = P[X_0 = q]$ in an order 1 MC (see definition 1); it defines the probability of the first $N-1$ outcomes of the process. The complete model consists in $|\mathcal{Y}|$ $N$-gram models; the parameters of the model associated to the class $y$ are superscripted with $y$:

- $p^y_{\mathtt{a} \mid h} = P[\mathtt{a} \mid h, y]$ for all $h \in \Sigma^{N-1}$ and $\mathtt{a} \in \Sigma$
- $p^y_v = P[v, y]$ for all $v \in \Sigma^{N-1}$

The parameters $p^y_{\mathtt{a} \mid h}$ encodes the conditional probabilities of the model while $p^y_v$ defines the initial probabilities of the model. Since sequences are assumed to have been independently drawn, the conditional log-likelihood of the complete training data $Tr$ is given by:

$$\mathrm{CLL}(Tr) = \sum_{(s,y) \in Tr} \log(P[y \mid s]) \tag{6.7}$$

The authors of [137] propose to use a gradient-ascent technique to maximize the conditional likelihood. In order to keep the parameters of the model in the range $[0, 1]$ and to enforce properness constraints, a new parametrization is used:

$$\begin{array}{lcll} p^y_{\mathtt{a} \mid h} & = & \dfrac{1}{Z_w} \exp(w^y_{\mathtt{a} \mid h}) & \text{with } Z_w = \sum_{\mathtt{a} \in \Sigma} \exp(w^y_{\mathtt{a} \mid h}) \\ p^y_v & = & \dfrac{1}{Z_u} \exp(u^y_v) & \text{with } Z_u = \sum_{v \in \Sigma^{N-1}, y \in \mathcal{Y}} \exp(u^y_v) \end{array} \tag{6.8}$$

where the new parameters $w^y_{\mathtt{a} \mid h}$ and $u^y_v$ are respectively related to the logarithm of the original parameters $p^y_{\mathtt{a} \mid h}$ and $p^y_v$. These new parameters are initialized according to the logarithm of their maximum likelihood estimates (see section 1.5). Then, they are iteratively updated according to the following gradient-based rules:

$$\begin{array}{lcl} w^y_{\mathtt{a} \mid h} & \leftarrow & w^y_{\mathtt{a} \mid h} + \alpha \, \dfrac{\partial \log \mathrm{CLL}(Tr)}{\partial w^y_{\mathtt{a} \mid h}} \\ u^y_v & \leftarrow & u^y_v + \alpha \, \dfrac{\partial \log \mathrm{CLL}(Tr)}{\partial u^y_v} \end{array} \tag{6.9}$$

where $\alpha$ is the learning rate. For the technical derivations of the partial derivatives, the reader is referred to the original paper [137]. Like the maximum likelihood estimates (see section 1.5), these partial derivatives rely on $N$-gram counts but they also include an additional term enforcing the true labels of sequences (provided in the training set) to have a larger posterior probability than the other labels. It is important to note that this learning technique is *not* guaranteed to find the global optimum of the conditional likelihood function whereas globally maximizing the classical likelihood is straightforward using formula provided in section 1.5. Once the model is estimated, new sequences are classified by assigning the most probable class: $y^* = \arg\max_{y' \in \mathcal{Y}} P[y \mid s]$. Experimental results show that discriminative $N$-grams offer slightly better performances than their generative counterpart but they are generally outperformed by kernel methods [137].

### 6.2.2 Kernel methods for sequence classification

Support Vector Machines (SVM) reviewed in chapter 3 can be applied to sequence classification problems by using a kernel computing similarities between sequences. Kernel methods are now considered as the state-of-the-art in supervised sequence classification. Many string or sequence kernels have been proposed over the last few years; each such kernel defines an embedding of the sequences into some relevant feature space where the maximal margin hyperplane is computed. This section reviews three important string kernels which are now considered as standard in the literature, namely the *spectrum kernel*, the *mismatch kernel* and the *subsequence kernel*. Several extensions of these kernels are presented in [118]. Kernels can also be constructed from generative models such as $N$-grams or HMMs. This is the objective of the so-called *marginalization* and *Fisher* kernels [118]. Marginalization kernels are used in section 7.4.2 to define an original kernel based on PH distribution. The Fisher kernel is briefly overviewed in the present section.

#### Spectrum kernel

The spectrum kernel proposed in [82] is one of the first kernels on sequences. It relies on the following principle: two sequences should be considered as similar if they share a large number of contiguous subsequences of length $p \in \mathbb{N}$. Therefore, a sequence $s$ is characterized by the histogram $\Phi^p(s)$ of frequencies of all contiguous subsequences of length $p$ called the *$p$-spectrum* of $s$. The $p$-spectrum kernel computes the dot product between two $p$-

spectra.

$$\begin{aligned} k_p(s,s') &= \langle \Phi^p(s), \Phi^p(s') \rangle \\ &= \sum_{v \in \Sigma^p} C_s(v)\ C_{s'}(v) \end{aligned} \tag{6.10}$$

where $C_s(v)$ denotes the number of occurrences of $v$ in $s$. For instance, let us consider the two following sequences defined on the alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}\}$: $s = \mathtt{abba}$ and $s' = \mathtt{baba}$. The 2-spectra for these sequences are respectively $\Phi^2(s) = \{(\mathtt{ab}, 1), (\mathtt{ba}, 1), (\mathtt{bb}, 1)\}$ and $\Phi^2(s') = \{(\mathtt{ab}, 1), (\mathtt{ba}, 2)\}$ where the (null) frequencies of non-occurring subsequences are not given. Performing the dot product between these feature vectors provides $k_2(s, s') = 3$. The spectrum kernel can be extended so as to consider contiguous subsequence *up to* length $p$. This kernel is known as the *blended* $p$-spectrum kernel. Since the frequencies of observed subsequences tend to decay exponentially with their length, one introduces weighted features in order to reinforce the influence of longer subsequences: $\Phi^{d,c}(s) = c^d \Phi^d(s)$, where $c \in \mathbb{R}^+$. The blended spectrum kernel is defined as follows:

$$\begin{aligned} k_{p,c}(s,s') &= \langle \Phi^{p,c}(s), \Phi^{p,c}(s') \rangle \\ &= \sum_{d=1}^{p} \sum_{v \in \Sigma^d} c^{2d}\ C_s(v)\ C_{s'}(v) \end{aligned} \tag{6.11}$$

The blended 2-spectra, with $c = 2$, of our example sequences are respectively $\Phi^{2,2}(s) = \{(\mathtt{a}, 4), (\mathtt{b}, 4),\ (\mathtt{ab}, 4), (\mathtt{ba}, 4), (\mathtt{bb}, 4)\}$ and $\Phi^{2,2}(s') = \{(\mathtt{a}, 4), (\mathtt{b}, 4),$ $(\mathtt{ab}, 4), (\mathtt{ba}, 8)\}$ and the related kernel evaluation is $k_2^2(s, s') = 80$. The spectrum kernel can be efficiently computed using a prefix tree data structure. The tree used to evaluate the 2-spectrum kernel on our example sequences is depicted in Figure 6.2. The nodes of the tree are labeled with their relative prefix. To each node $q$ are associated two lists $L_s$ and $L_{s'}$. In the root node $\varepsilon$, $L_s$ and $L_{s'}$ are respectively initialized with the frequencies of subsequences of length $p$ in $s$ and $s'$. These subsequences are filtered down the tree and the list $L_s/L_{s'}$ associated to a node $q$ contains the frequencies of all subsequences in $s/s'$ starting with $q$. Given a node $q$, let $n^s(q)$ and $n^{s'}(q)$ respectively denote the summed frequencies of subsequences in $L_s$ and $L_{s'}$. Summing the products $[n^s(q)\ n^{s'}(q)]$ over all the leaves $q$ in the tree provides the desired kernel evaluation $k_p(s, s')$. This computation has a time complexity $\mathcal{O}(p(|s| + |s'|))$. If $\mathrm{dp}(q)$ denotes the depth of the node $q$ in the tree[2], the blended spectrum kernel is obtained by summing $[c^{2\mathrm{dp}(q)} n^s(q)\ n^{s'}(q)]$ over all nodes $q$ in the tree but the root. Such a computation has the same time complexity as the non-blended spectrum kernel.

[2]The depth of the root node is defined as zero.

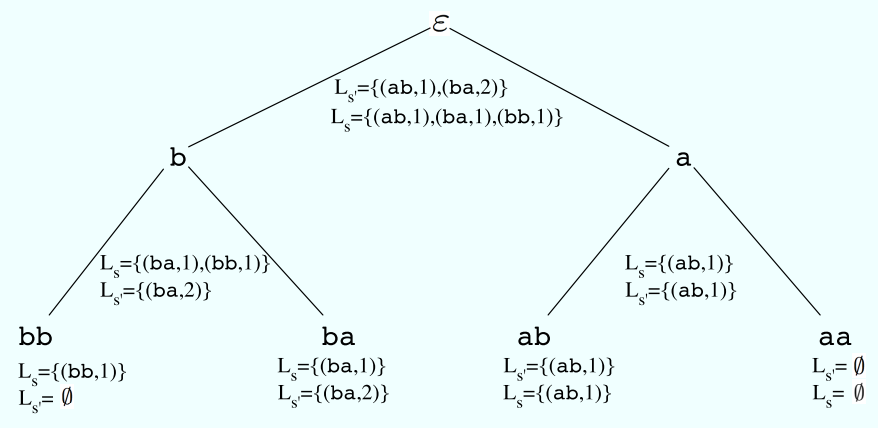



Figure 6.2: The prefix tree used to compute the 2-spectrum kernel applied on the sequences $s = \texttt{abba}$ and $s' = \texttt{baba}$.

We now provide an interesting link between SVM along with the spectrum kernel and classical MC. To the best of our knowledge, this relation has not been provided in the literature. Let us consider an order $p-1$ MC denoted by $T$ and a sequence $s = s_0 \dots s_l$. Ignoring[3] the initial probability, the likelihood of $s$ with respect to $T$ is provided by

$$\begin{aligned} P[s \mid T] &= \prod_{t=p-1}^{l} P[s_t \mid s_{t-p-1:t-1}] \\ &= \prod_{hx \in \Sigma^p} (P[x \mid h])^{C_s(hx)} \end{aligned} \tag{6.12}$$

where $x$ is an individual symbol and $h$ is a subsequence of length $p-1$. Consequently, the log-likelihood of $s$ with respect to $T$ is

$$\log(P[s \mid T]) = \sum_{hx \in \Sigma^p} C_s(hx) \log(P[x \mid h]) \tag{6.13}$$

If one parametrizes the conditional log-probabilities as $w_{hx} = \log(P[x \mid h])$, the log-likelihood becomes:

$$\begin{aligned} \log(P[s \mid T]) &= \sum_{v \in \Sigma^p} C_s(v)\, w_v \\ &= \langle \boldsymbol{w}, \Phi^p(s) \rangle \end{aligned} \tag{6.14}$$

where $\boldsymbol{w}$ is a $|\Sigma|^p$-dimensional vector gathering the conditional log-probabilities. This is exactly the functional form of the decision function of a SVM based

[3]The initial probability can be taken into account by adding $p-1$ silent symbols $\varepsilon$ at the beginning of sequences.

on the spectrum kernel, without considering the bias term $b$ (see section 3.3.1). Consequently, learning a SVM along with the spectrum kernel amounts to learn a standard MC in a discriminative fashion. We shall point out two remarks: (i) the model learned by SVM needs not satisfy stochastic or properness constraints, therefore $\langle \boldsymbol{w}, \Phi^p(s) \rangle$ should be interpreted as a score rather than as a log-probability, and (ii) the additional bias term $b$ in the SVM decision function acts as a threshold for classifying sequences according to their score. In contrast with the method optimizing the conditional likelihood of MC (see section 6.2.1), SVM globally optimizes its objective function. Finally, using blended spectrum features is equivalent to learning a mixture of increasing orders MC where the weights are defined according to the blending weight parameter $c$.

### Mismatch kernel

The mismatch kernel [81] generalizes the spectrum kernel by comparing the frequencies of subsequences which are not exactly the same. Let $v$ and $v'$ denote two subsequences defined over an alphabet $\Sigma$ and such that $|v| = |v'| = p$. The number of mismatches $d(v, v')$ is the number of symbols to substitute in one of these sequences to obtain the other. Given a subsequence $v$ of length $p$ and a number $m$ of allowed mismatches, the *mismatch feature* with respect to a sequence $s$ is defined as follows:

$$\Phi_v^{p,m}(s) = \sum_{\{v' \in \Sigma^p \mid d(v,v') \leq m\}} C_s(v') \tag{6.15}$$

That is, it counts the occurrences of subsequences of length $p$ differing by at most $m$ symbol(s) from $v$. Consequently, a sequence $s$ is defined by a $|\Sigma|^p$-dimensional vector $\Phi^{p,m}(s)$ such that the entry corresponding to a subsequence $v$ is $\Phi_v^{p,m}(s)$. The mismatch kernel is defined as the dot product between such feature vectors:

$$\begin{aligned} k_{p,m}(s, s') &= \langle \Phi^{p,m}(s), \Phi^{p,m}(s') \rangle \\ &= \sum_{v \in \Sigma^p} \sum_{\{v' \in \Sigma^p \mid d(v,v') \leq m\}} C_s(v') C_{s'}(v') \end{aligned} \tag{6.16}$$

For instance, let us compute the mismatch feature with $p = 3$, $m = 1$ and $v = \texttt{aba}$ of the sequence $s = \texttt{abba}$. The frequencies of all subsequences of length 3 obtained by changing at most one symbol in $v$ are the following: (aba,0), (bba,1), (abb,1) and (aaa,0). Consequently, $\Phi_v^{p,m}(s) = 2$. Again, the mismatch kernel is efficiently computed using a prefix tree data structure. The tree used to evaluate the mismatch kernel with $p = 3$ and $m = 1$ on our example sequences is displayed in Figure 6.3. It is constructed in a similar

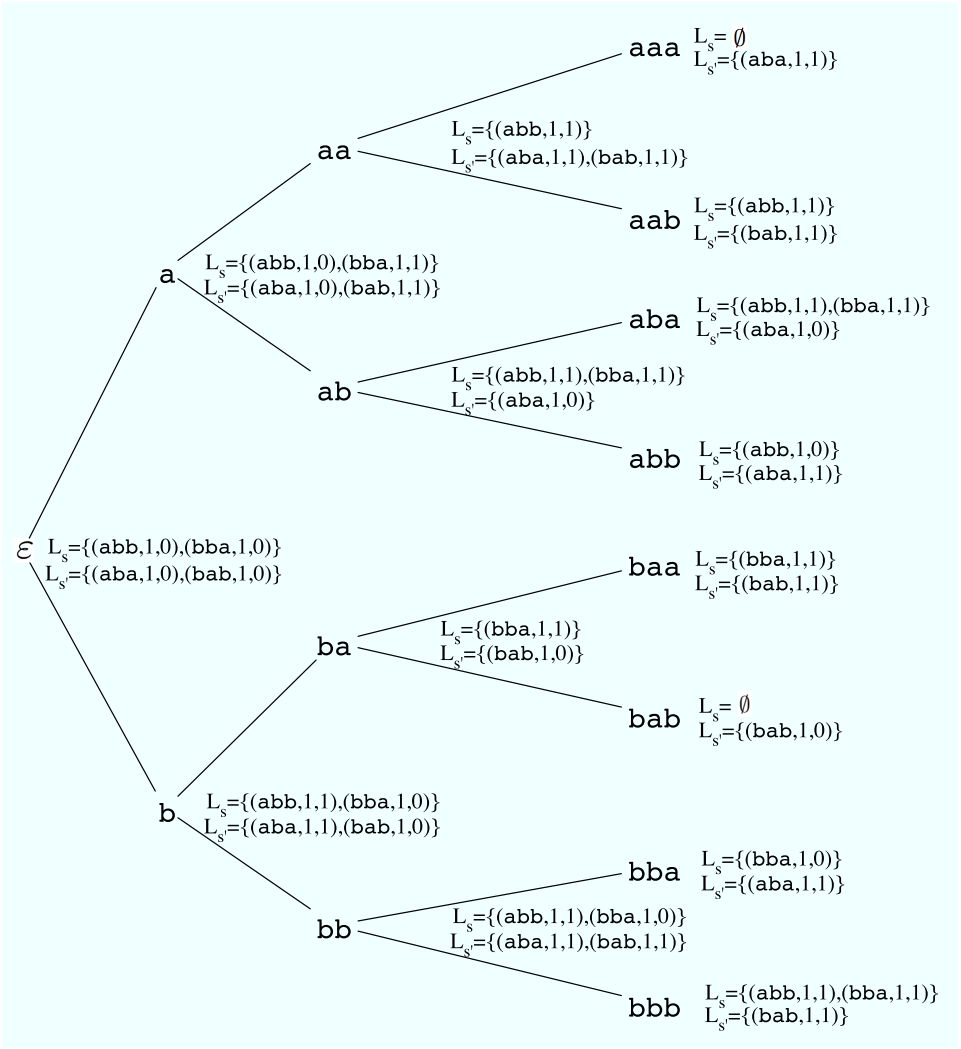



Figure 6.3: The prefix tree used to compute the mismatch kernel with $p = 3$ and $m = 1$ applied on the sequences $s =$ `abba` and $s' =$ `baba`.

way as the spectrum kernel except the elements in the list $L_s/L_{s'}$ attached to the nodes $q$ are now 3-tuples $(v, i, j)$ where $v$ is a subsequence of length $p$, $i$ is the frequency of $v$ in $s/s'$ and $j$ is the number of symbol substitutions required for $q$ to be a prefix of $s/s'$. The subsequences of length $p$ present in $s$ and $s'$ are filtered down the tree until reaching a leaf or exceeding the allowed number $m$ of mismatches. Unlike for the spectrum kernel, a subsequence $v$ can traverse the prefix tree along several paths, each one corresponding to particular symbol substitutions. By construction, at a leaf $q$, the summed frequencies $n^s(q)$ and $n^{s'}(q)$ are respectively equal to $\Phi_q^{p,m}(s)$ and $\Phi_q^{p,m}(s')$. Consequently, summing the products $[n^s(q)\ n^{s'}(q)]$ over all the leaves $q$ in the tree provides the desired kernel evaluation $k_{p,m}(s, s')$. In our example, the kernel evaluation is $k_{p,m}(s, s') = 8$. This computation has a time complexity $\mathcal{O}(p^{m+1}|\Sigma|^m(|s| + |t|))$. The mismatch kernel is clearly more expensive to compute than the spectrum kernel. This additional complexity can however be controlled by restricting the number $m$ of allowed mismatches.

**Subsequence kernel**

Another extension of the (blended) spectrum kernel considers features made of possibly non-contiguous subsequences. This kernel is called here the *subsequence kernel* [88]. To define these extended features, we shall introduce some new notations. Given a sequence $s = s_0 \dots s_l$, $\mathbf{i} = \{i_1, \dots, i_k\}$ denotes a list of k strictly increasing positions or indices in $s$. The notation $s(\mathbf{i})$ stands for the subsequence defined by considering only the positions $\mathbf{i}$ in $s$. For instance, let us consider the sequence $s = \texttt{anaconda}$ and the set of positions $\mathbf{i} = \{0, 1, 5, 7\}$. The subsequence of $s$ defined by $\mathbf{i}$ is $s(\mathbf{i}) = \texttt{anna}$. The feature space induced by the subsequence kernel is indexed by all the subsequences $v \in \Sigma^*$. The value of the feature associated to a subsequence $v$ is equal to the number of times $v$ is matched as a subsequence of $s$:

$$\Phi_{sub}^v(s) = |\{\mathbf{i} \mid s(\mathbf{i}) = v\}| \tag{6.17}$$

Note that the empty sequence $\varepsilon$ is assumed to be matched one time in any input sequence. Gathering features associated to all subsequences $v \in \Sigma^*$ gives rise to a feature vector $\Phi_{sub}(s)$. The subsequence kernel is defined as the dot product between such feature vectors:

$$\begin{aligned} k_{sub}(s, s') &= \langle \Phi_{sub}(s), \Phi_{sub}(s') \rangle \\ &= |\{(\mathbf{i}, \mathbf{j}) \mid s(\mathbf{i}) = s'(\mathbf{j})\}| \end{aligned} \tag{6.18}$$

where $\mathbf{j}$ is set of positions in $s'$. To illustrate this point, let us compute the kernel matrix for the set of sequences $S = \{\texttt{tac}, \texttt{cat}, \texttt{acc}\}$. Table 6.1 shows the number of matchings of all subsequences in $S$.

| | $\varepsilon$ | a | c | t | ac | at | ca | cc | ct | ta | tc | acc | cat | tac |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| tac | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| cat | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| acc | 1 | 1 | 2 | 0 | 2 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |

Table 6.1: The numbers of subsequence matchings in the input sequences $\{\texttt{tac}, \texttt{cat}, \texttt{acc}\}$.

Computing the dot product between these feature vectors (which are the rows of Table 6.1) provides the following kernel matrix $K$:

| $K$ | tac | cat | acc |
|---|---|---|---|
| tac | 8 | 4 | 6 |
| cat | 4 | 8 | 4 |
| acc | 6 | 4 | 12 |

Note that the diagonal elements, computing the sequence self-similarities, are not all the same, even if the considered input sequences have the same length. This can be avoided by normalizing the kernel matrix as detailed in [118].

Computing the subsequence kernel explicitly in the feature space, as we did in the previous example, becomes computationally intractable while applied to longer sequences. Indeed, even if we use a sparse representation of the feature vectors, the number of non-zero entries grows exponentially fast with the length of the subsequences (a bound on this number is given in [118]). Fortunately, there is an efficient algorithm, based on dynamic programming, for computing the subsequence kernel. Considering two input sequences $s$ and $s'$, this procedure relies on the following recurrence:

$$\text{(Base case)} \quad k(s, \varepsilon) = 1 \tag{6.19}$$

$$\text{(Induction)} \quad k(u\texttt{a}, s') = k(u, s') + \sum_{\{i \mid s'_i = \texttt{a}\}} k(u, s'_{0:i-1}) \tag{6.20}$$

The base case simply states that the empty sequence $\varepsilon$ is matched exactly one time in $s'$. The induction step decomposes the input sequence $s$ into a prefix $u$ consisting of all the symbols in $s$ but the last, and the final symbol denoted here by $\texttt{a}$. Counting the number of common subsequences in $s$ and $s'$ is split into two parts. The first term of equation (6.20) counts the number of common subsequences that do *not use* the last symbol of $s$. The second term of equation (6.20) counts the number of common subsequence while *using*

the last symbol a of $s$. For the latter case, one counts the number of common subsequences in $u$ and any prefix of $s'$ followed by an a. This recursion is efficiently computed using a $|s| \times |s'|$ dynamic programming array (for details about this computation see [88] or [118]). It has a time complexity $\mathcal{O}(|s||s'|)$. Extensions of the subsequence kernel consider fixed length subsequences and gap-weighted subsequences [118] to penalize strongly non-contiguous features.

**Fisher kernel**

The Fisher kernel [64, 114] is an interesting technique for incorporating generative information into a discriminative classifier. Unlike marginalization kernels presented in section 7.4.2, the Fisher kernel is based on a single generative model $M_\rho$ which is smoothly parametrized by a set of parameters $\rho = \{\rho_1, \ldots, \rho_k\}$. For instance, in the context of sequence classification, the generative model could be a HMM parametrized by its initial, transition and emission probabilities. The generative model is generally learned from training data but it can also be designed by experts of the application domain. In a nutshell, the Fisher kernel compares two instances $s^1$ and $s^2$ according to how they are generated by $M_\rho$. That is, $s^1$ and $s^2$ are similar if their internal representations in $M_\rho$ are close. To define such a similarity measure, one computes separately for $s^1$ and $s^2$ how the parameters $\rho$ should be updated to increase the sequence likelihood. This gives rise to two update vectors which are compared using a dot product, providing the desired similarity measure. We now give some details about this computation.

Let $P[s \mid \rho]$ denotes the likelihood of the sequence $s$ with respect to the model $M_\rho$. We will compute, using a gradient-based strategy, how the model parameters should be updated to increase the (log-)likelihood of $s$. Note that HMMs can be estimated using such gradient-based technique or using the Baum-Welch algorithm which can be thought of as a gradient ascent where the learning rate is automatically adapted (see [113]). The $k$-dimensional gradient vector of the likelihood function for an instance $s$ with respect to $\rho$ is called the *Fisher score*:

$$\boldsymbol{g}(s, \rho) = \{\partial_{\rho_i} P[s \mid \rho]\}_{i=1}^{k} \tag{6.21}$$

The *Fisher information matrix*, is built from such feature vectors as follows:

$$I_\rho = \mathbb{E}\ \left[\boldsymbol{g}(s, \rho)\ \boldsymbol{g}(s, \rho)^T\right] \tag{6.22}$$

where the expectation is over the complete data distribution $P[s]$ which is unavailable in practice. Instead, one can compute the *empirical Fisher*

*information matrix* from the training data as follows :

$$\hat{I}_M = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{g}(s^i, \rho)\ \boldsymbol{g}(s^i, \rho)^T \tag{6.23}$$

In other words, the empirical Fisher information matrix is the covariance matrix between Fisher feature vectors. Unfortunately, this empirical expectation can amplify particularities of the training data which are not representative of the complete data distribution, leading to noisy covariances. The Fisher information matrix (or its empirical estimation) is used to compute a modified dot product between two Fisher feature vectors. This dot product defines the *invariant Fisher kernel*:

$$k_\rho^{inv}(s, s') = \boldsymbol{g}(s, \rho)^T\ I_\rho^{-1} \boldsymbol{g}(s', \rho) \tag{6.24}$$

This kernel is said to be *invariant* as it is not affected by any smooth and invertible reparameterization of the model parameters (for instance, using a logarithm function). This property is formally shown in [118]. This is interesting when the parametrization of the model is somewhat arbitrary. Alternatively, one can use the *practical Fisher kernel*:

$$k_\rho(s, s') = \boldsymbol{g}(s, \rho)^T\ \boldsymbol{g}(s', \rho) \tag{6.25}$$

The practical Fisher kernel is not invariant to parameter reparameterization however it does not suffer from a noisy information matrix estimation. Technical computations of HMM log-likelihood partial derivatives can be found in [118]. Evaluating these derivatives for a sequence $s$ rely on the forward and backward lattices used in the Baum-Welch algorithm (see section 2.3.3). The time complexity for computing the Fisher feature vector associated to $s$ is $\mathcal{O}(|s|k)$. Note that the number $k$ of model parameters is $\mathcal{O}(n_t + n_s|\Sigma|)$ where $n_s$ and $n_t$ are respectively the number of states and of transitions in the model, and $|\Sigma|$ is the size of the alphabet. Finally, both Fisher kernels, applied to a pair of sequences $(s, s')$, have a time complexity $\mathcal{O}((|s| + |s'|)k + k^2)$.

## 6.3 Summary and discussion

In this chapter we have provided an overview of some sequence classification techniques. The latter are dichotomized here into generative and discriminative approaches. On the one hand, generative methods model the joint distribution of the sequences and their categories. That is, one models the generative process of the training data. Classification of new sequences is performed via a maximum likelihood or a maximum *a posterior* decision

rule. On the other hand, discriminative techniques directly focus on the decision boundary, that is on what is most important to perform classification. It is well-known that discriminative techniques often offer better classification results than their generative counterparts. Indeed, models resulting from such techniques only include relevant informations to discriminate sequences, not to generate them. Besides, according to Vapnik, one should not solve a more general problem than the considered task. That is, one should not learn how sequences are generated, *i.e.* the generative process, to discriminate them. Nevertheless, optimizing a discriminative function such as the conditional likelihood is often much harder than dealing with the classical sequence likelihood (for instance, see section 4.3.1). Moreover, hybrid techniques incorporating generative and discriminative components have also provided very good results in practice. Consequently, these two approaches are complementary.

In the context of generative approaches, the Naive Bayes classifier is considered as a good base line technique. Such a classifier essentially relies on the frequencies of individual symbols within each class. That is, it assumes that symbols in sequences (or text) are generated independently and consequently a random permutation of symbol positions has no effect on such a classifier. Despite this rather crude assumption, good results are obtained in practice, especially in text classification. Naive Bayes can be extended by using $N$-gram frequencies instead of frequencies of individual symbols. Doing so models the symbol co-locations within a window of length $N$. However, such models are more difficult to estimate accurately as the potential number of distinct $N$-grams grows exponentially fast with the window length $N$. To face this issue, robust estimation techniques, based on back-off smoothing, have been presented in section 1.5. The smoothed models actually define variable order Markov chains (MC) the order of which ranges from 0 to $N-1$.

Other estimation techniques defining variable order MC are discussed and compared in [9]. We have focused on a technique based on the well-known compression algorithm LZ78. This method has been shown to consistently outperform the competing approaches while applied to sequence classification tasks. Interestingly, there is a close relation between compression and prediction since, according to Shannon, data are encoded more concisely when the generative distribution of the data (or a model of this distribution) is known. LZ78 compresses a message (*e.g.* a file) by replacing portions of data with references to previously encountered fragments stored in a lexicon. Such a lexicon is efficiently represented via a prefix tree from which conditional probabilities can be computed. These conditional probabilities are crudely estimated but are sufficient for compression

purpose. Extensions of LZ78 [97] improve these noisy estimations in very simple ways. The resulting variable order MC have been shown to produce very good results in biological sequence classification [9].

Alternatively, one can generalize Naive Bayes using even more powerful models such as Hidden Markov Models. These models are however harder to train since the structure or topology has to be defined in addition to parameter estimation. This question has been thoroughly discussed in chapter 4 whereas chapter 5 has proposed a new structural induction technique relying on First Passage Times (FPT).

Discriminative approaches learn model to be specifically used in sequence classification. The considered models can be probabilistic such as MC and HMMs which are trained discriminatively, or discriminative functions such as a hyperplane in a feature space. Several discriminative criteria for learning generative models in a discriminative fashion have been mentioned in section 4.3. The conditional likelihood is perhaps the most popular criterion towards this objective. We have reviewed the discriminative counterparts of $N$-gram classifiers, that is MC optimizing the conditional likelihood. Such models are estimated using a gradient-based technique which is not guaranteed to find the global optimum of the conditional likelihood function. Practical results show that discriminatively trained MC slightly outperform their generative counterparts. However, they are generally consistently outperformed by kernel methods.

Kernel methods are now considered as the state-of-the-art in sequence classification tasks. They basically consist of a Support Vector Machine (SVM) classifier along with a string kernel embedding the sequences into some relevant feature space. Three important string kernels have been reviewed: (i) the (blended) spectrum kernel (ii) the mismatch kernel and (iii) the subsequence kernel. All these kernels compute similarities according to the number of subsequences shared by two input sequences. The resulting feature space is indexed by all the considered subsequences. The spectrum kernel considers subsequences of fixed length whereas the blended spectrum kernel deals with subsequences up to some prescribed length. We have provide a new theoretical link between these two kernels and classical MC. Roughly speaking, training a SVM based on the spectrum kernel amounts to discriminatively learn a finite order MC. Using the blended alternative estimates a mixture of increasing order MC. Unlike the technique optimizing the conditional likelihood of MC, SVM is guaranteed to maximize its discriminative criterion which is a trade-off between the margin maximization and the minimization of a bound on the empirical risk (see section 3.3.3). The mismatch kernel generalizes the spectrum kernel by comparing

the frequencies of subsequences which are not matching perfectly. This is interesting in the case of noisy sequences where some symbols might be ambiguous. Unlike the two preceding kernels, the subsequence kernel considers possibly non-contiguous subsequences. Interestingly, these features are no longer local, as subsequences can potentially be spread in the whole input sequences.

Hybrid classification methods coupling generative and discriminative components can be defined by incorporating generative information in the design of kernels. The marginalization and the Fisher kernels are two approaches towards this objective. A marginalization kernel builds a feature space from a collection of generative models. Each axis of such a feature space corresponds to the likelihood of the data instance with respect to the associated generative model. This technique is used in section 7.4.2 to define a new kernel based on phase-type (PH) distributions. In contrast with the marginalization kernel, the Fisher kernel is based on a single generative model. It compares two instances according to how they are generated by the model. For each training instance, one computes, via the gradient of the likelihood function, how the model should be updated to increase the likelihood. Two sequences are similar if the model should be updated in the same way. In the context of sequence classification, the considered generative model is typically a MC or a HMM.

In chapter 7, we propose a novel approach to perform sequence classification. A completely different point of view is adopted for characterizing the input sequences. Rather than relying on the frequencies of subsequences, we focus on the temporal dynamics between occurrences of such subsequences. In other words, one computes First Passage Time (FPT) features from the input sequence. All the approaches presented in the current chapter, except the subsequence kernel, focus on the local dynamics in sequences. This is somewhat restrictive, as two distinct processes having the same local dynamics might have a very different FPT dynamics. This is illustrated in section 5.4.1 where a target process with a complex temporal dynamics, including long-term dependencies, is modeled with a finite order MC. While both processes share the same local dynamics, Figure 5.9 shows that their non-local dynamics are very different. Using FPT features allows one to discriminate sequences drawn from two such processes. These features are modeled by phase-type (PH) distributions which also characterize the FPT dynamics in HMMs and POMMs (see section 5.3). Doing so, one includes a part of the powerful expressiveness of HMMs into classification techniques. Two classifiers are proposed: (i) a generative classifier using PH distributions to compute the sequence likelihood (ii) a hybrid approach consisting of a SVM with a marginalization kernel based on PH distributions. Additionally, a

feature selection procedure is proposed to select the most informative FPT features.

# Chapter 7

# Sequence classification using First Passage Times

*This chapter presents an original approach to the discrete sequence classification problem. Classification techniques presented in chapter 6 are based on the frequencies of particular subsequences (e.g. N-grams or non-contiguous subsequences of a given length) in sequences. In contrast with these methods, our approach characterizes sequences in the temporal domain. That is, we focus on the time (i.e. the number of steps) between occurrences of subsequences rather than on the frequencies of these subsequences. More precisely, given a pair of subsequences* $(v^1, v^2)$*, called here a* feature *of the sequence to be classified, we are looking at the number of steps taken to observe the next occurrence of* $v^2$ *after having observed* $v^1$*. The distribution of these measures forms the First Passage Time (FPT) dynamics of a sequential process with respect to the feature* $(v^1, v^2)$*. Importantly, FPT features do not only focus on the local dynamics in sequences as there is no a priori fixed maximum time (i.e. number of steps) between two events. Such features are therefore well-suited to capture long-term dependencies or complex time-dependent dynamics in sequences. The induction algorithm* POMMStruct *presented in section 5.5 builds a Partially Observable Markov Model (POMM) from FPT between individual symbols observed in the learning sample. In contrast with* POMMStruct*, the FPT considered here are measured between occurrences of subsequences up to a prescribed length rather than between individual symbols. This offers a more powerful modeling as such features can capture dependencies between aggregated events (i.e. subsequences) in sequences.*

*The purpose of our approach is to exploit the different FPT dynamics between the classes to perform sequence classification. Since the number of features can be potentially large, only a restricted number of the ob-*

*served features are considered. Feature selection is performed using the Jensen-Shannon (JS) divergence [86]. This function measures whether several samples, defined by their empirical distributions, are drawn from different source distributions. Given an observed feature* $(v^1, v^2)$*, the empirical FPT distribution is estimated for each class. The JS divergence is then applied to rank the considered features. The features offering the largest JS divergence between their class conditional distributions are kept for the classification process. Once the features have been selected, the associated FPT are modeled with general*[1] *phase-type (PH) distributions (see section 1.4). PH distributions form a broad class of distributions that generalize the family of negative binomial distributions. Moreover, we have shown in section 5.3 that the FPT between symbols in POMMs (or equivalently in HMMs) are drawn from PH distributions. The* POMMStruct *algorithm (see section 5.5) learns the structure and the parameters of POMMs by fitting these PH distributions from data. In contrast with this induction technique, each PH distribution is fitted here on a separate absorbing MC. That is, for each selected feature* $(v^1, v^2)$*, an absorbing MC is defined, the parameters of which are estimated so as to fit the observed FPT from* $v^1$ *to* $v^2$*. To do so, we propose a novel EM-based algorithm for fitting discrete PH distributions called* PHit*. This technique can be considered as an adaptation to discrete distributions of the work of Asmussen and Olsson [5], which handles continuous PH distributions. It can also be considered as a simplification of the* POMMPHit *algorithm (see section 5.5.2) which fits several PH distributions with a single model. Modeling the FPT dynamics with PH distributions controls the generalization, that is, the probability mass given to unseen events, by tuning the number of phases (see section 1.4). In this sense, the PH modeling can be thought of as a smoothing technique of the empirical FPT distributions observed in the sequences. The estimated PH distributions are used to solve sequence classification problems. Our first classification technique is purely generative: it consists of a maximum a posteriori (MAP) classifier where the likelihood is computed using PH distributions. The second classification technique is hybrid, i.e. it combines generative and discriminative components, as it consists of a Support Vector Machine (SVM) classifier applied in a feature space built from PH distributions. This feature space is induced by a novel kernel on sequences called the* PH *kernel.*

*This chapter is organized as follows. Section 7.1 presents the general strategy to classify sequences using FPT. Section 7.2 details the feature se-*

[1] A general phase-type distribution is a PH distribution associated to an absorbing MC with no particular structure.

*lection procedure to extract relevant FPT pairs. Section 7.3 presents the* PHit *algorithm for fitting a PH distribution from data using an absorbing MC. Section 7.4 presents a MAP classifier and a string kernel; both are based on PH distributions. Finally, section 7.5 shows experimental results obtained with the proposed methods applied to biological sequence classification problems.*

## 7.1 Classification using first passage times

This section presents a general scheme to perform sequence classification using FPT. This scheme is illustrated in Figure 7.1 . The different steps are briefly sketched here; they are further detailed in sections 7.2, 7.3 and 7.4. The inputs consist of a training set containing $n$ labeled sequences $Tr = \{(s^1, y_1), \ldots, (s^n, y_n)\}$ where $s^i \in \Sigma^*$ is a sequence and $y_i \in \mathcal{Y}$ is its label or class, a number $N$ defining the maximal subsequence length considered in the features, a number $p$ of features to extract and a number $n_s$ specifying the number of states[2] to use for fitting the selected FPT. Our classification approach consists of 3 steps: (i) feature extraction and selection, (ii) fitting the selected features with PH distributions, and (iii) building a classifier. First, the FPT are extracted separately for each class from the training sequences (see definition 41). The feature selection procedure aims at detecting the features $(v^1, v^2)$ for which the FPT dynamics strongly varies among the classes. To do so, features are ranked using a procedure detailed in section 7.2 and the $p$ highest ranked features are kept for the classification process. The set $\mathcal{P}$ denotes the set of selected features. Next, the FPT associated to each pair $(v^1, v^2) \in \mathcal{P}$ are fitted using a PH distribution containing $n_s - 1$ phases. More precisely, given a pair $(v^1, v^2) \in \mathcal{P}$, a PH distribution is estimated for each class $y \in \mathcal{Y}$ using the FPT extracted from the sequences labeled with $y$. The fitting procedure, called PHit, is detailed in section 7.3. Once the PH distributions have been fitted, a classifier based on these distributions is built. Two classifiers are proposed: (i) a MAP classifier using the PH distributions as a generative model and (ii) a SVM classifier along with a kernel based on PH distributions. These classifiers are further detailed in section 7.4.

## 7.2 Feature extraction and selection

The features extracted from the training sequences are the First Passage Times (FPT) between occurrences of subsequences.

[2]The number of phases of a PH distribution associated to a reduced absorbing MC with $n_s$ states is $n_s - 1$.

**Inputs:**
- A training set $Tr = \{(s^1, y_1), \ldots, (s^n, y_n)\}$
- A maximal subsequence length $N$
- A number $p$ of features
- A number $n_s$ of states

**Output:**
- A classifier

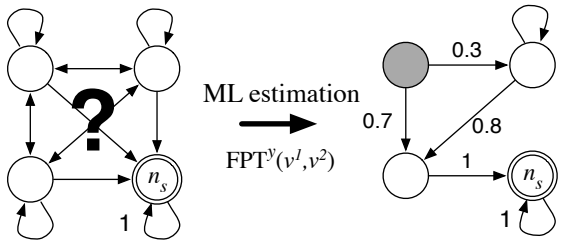
**1. Feature extraction and selection**

| | Feature | $\text{FPT}^{y_1}$ | ... | $\text{FPT}^{y_{\lvert\mathcal{Y}\rvert}}$ | Score |
|---|---|---|---|---|---|
| 1 | (ba,b) | {(1,10), ...,(50,2)} | ... | {(1,2),..., (30,1)} | 0.6 |
| 2 | (a,bb) | {(1,28), ...,(20,2)} | ... | {(1,16),..., (15,2)} | 0.5 |
| | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| $p$ | (bc,ba) | {(1,10), ...,(25,2)} | ... | {(1,9),..., (24,3)} | 0.1 |
| | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

$\mathcal{P} \leftarrow \{(\texttt{ba}, \texttt{b}), (\texttt{a}, \texttt{bb}), \ldots, (\texttt{bc}, \texttt{ba})\}$

**2. Fitting PH distributions**

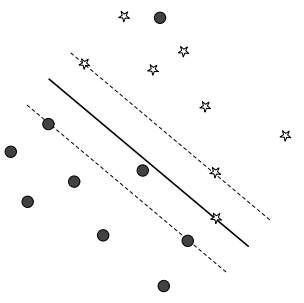
**for** $(v^1, v^2) \in \mathcal{P}$ **do**
  **for** $y \in \mathcal{Y}$ **do**




**3. Building classifier**

$$P[y \mid s] = \frac{P[s \mid y]P[y]}{\sum_{y' \in \mathcal{Y}} P[s \mid y']P[y']}$$

*Maximum a posteriori classifier*



*SVM in a PH feature space*

Figure 7.1: The general scheme used to learn a classifier from first passage times.

**Definition 41** *Given a sequence $s$ defined on an alphabet $\Sigma$ and two subsequences $v^1, v^2 \in \Sigma^+$. For each occurrence of $v^1$ in $s$, the first passage time to $v^2$ is defined as the finite number of steps taken for observing the next occurrence of $v^2$. $\mathrm{FPT}_s(v^1, v^2)$ denotes the first passage times to $v^2$ for all occurrences of $v^1$ in $s$. It is represented by a set of pairs $\{(z_1, c_1), \ldots, (z_l, c_l)\}$ where $z_i$ denotes a passage time and $c_i$ is the frequency of $z_i$ in $s$.*

For instance, let us consider the sequence $s = \texttt{aababbabba}$ defined on the alphabet $\Sigma = \{a, b\}$ and the events $v^1 = \texttt{ab}$ and $v^2 = \texttt{ba}$. The value of the feature $(\texttt{ab}, \texttt{ba})$ is $\mathrm{FPT}_s(\texttt{ab}, \texttt{ba}) = \{(3, 1), (1, 2)\}$. Note that the step count starts after the last character of $v^1$ and it does not take the length of $v^2$ into account. Definition 41 can be readily extended to apply to a set of sequences rather than to an individual sequence. This can be done by cumulating the frequencies of the passage times over all the sequences in the set. The FPT are extracted separately for each class $y \in \mathcal{Y}$ such that $\mathrm{FPT}^y(v^1, v^2)$ denotes the FPT from $v^1$ to $v^2$ computed from all the sequences labeled with $y$. The potential number of features is bounded by $(\sum_{i=1}^{N} |\Sigma|^i)^2 \in \mathcal{O}(|\Sigma|^{2N})$, where $|\Sigma|$ denotes the alphabet size. An alternative upper bound is $n.L^2.N^2$, where $L$ is the length of the longest sequence in the training set which contains $n$ sequences, and $N$ is the input parameter defining the maximal subsequence length considered in the features. Even if the number of features observed in practice is often below these bounds, it is crucial to select a reasonably small subset of features to keep the computational run-time tractable. Indeed, the number of features directly controls the number of PH distributions to fit in the second step of the classification scheme (see Figure 7.1).

Since our classification methods rely on the difference between the class conditional FPT dynamics, the proposed feature selection procedure extracts the features for which the empirical FPT distribution differs the most among the classes. To do so, a generalization of the Jensen-Shannon (JS) divergence is used [86]. The JS divergence has already been used to perform feature selection in the context of POMM induction (see equation (5.8)). The generalized version considered here can apply to more than two distributions. It is defined as follows

$$D_{JS}^{\mathrm{gen}}(P_1, \ldots, P_r) = H\left(\sum_{i=1}^{r} c_i P_i\right) - \sum_{i=1}^{r} c_i H(P_i) \tag{7.1}$$

where $P_1, \ldots, P_r$ are distributions defined on a set $\Omega$ of discrete events, $c_1, \ldots, c_r$ are strictly positive weights summing up to one and $H(P) = -\sum_{e \in \Omega} P[e] \log P[e]$ is the Shannon entropy. A larger JS divergence between the empirical distributions of various samples indicates that they are

more likely to have been drawn from different source distributions (*i.e.* from different classes). The JS divergence is a concave function and has several advantages over the Kullback-Leibler (KL) divergence [86], a classical measure to compare two distributions: (i) it can be applied to more than two distributions, (ii) the relative importance of the distributions can be parametrized with weights (in our experiments, uniform weights are used), (iii) it can be thought of as a symmetrized and smoothed (it is relative to the mean of the distributions) variant of the KL divergence, (iv) when applied to two distributions, the square root of the JS divergence enjoys the properties of a true distance metric. For each class $y \in \mathcal{Y}$, the *empirical FPT distribution* associated to a feature $(v^1, v^2)$, denoted by $\widehat{\mathrm{FPT}}^{y}(v^1, v^2)$, is obtained by computing the relative frequency of each distinct passage time from $v^1$ to $v^2$ in the sequences labeled with $y$. The score of a feature $(v^1, v^2)$ is defined as the JS divergence between the empirical FPT distributions of each class, weighted by the prior probability of the feature:

$$\mathrm{Score}(v^1, v^2) = P[(v^1, v^2)]\; D_{JS}^{\mathrm{gen}}(\widehat{\mathrm{FPT}}^{y_1}(v^1, v^2), \ldots, \widehat{\mathrm{FPT}}^{y_{|\mathcal{Y}|}}(v^1, v^2)) \quad (7.2)$$

where the prior probability of a feature $(v^1, v^2)$ is defined as

$$P[(v^1, v^2)] = \frac{\mathrm{count}(v^1, v^2)}{\sum_{v^3, v^4 \in \Sigma^{\leq N}} \mathrm{count}(v^3, v^4)} \quad (7.3)$$

such that $\mathrm{count}(v^1, v^2)$ is the number of times a string $v^1$ is followed (without overlap) by a string $v^2$ in the training set and $\Sigma^{\leq N}$ is the set of non-empty strings up to length $N$ defined on the alphabet $\Sigma$. The features are ranked with respect to their score and the $p$ highest ranked features are kept for the classification process, where $p$ is an input parameter. The selected features are denoted by the set $\mathcal{P}$. The computation of the empirical FPT distributions for every features made of subsequences of length up to $N$ can be done in time complexity $\mathcal{O}(n.L^2.N^2)$. Given one feature, the time complexity of the JS divergence computation is $\mathcal{O}(|\mathcal{Y}|.L)$ since the maximum value of the distributions support is bounded by $L$. The ranking of the features is done in time complexity $\mathcal{O}(R\ |\mathcal{Y}|\ L + R \log R)$ where $R = nL^2N^2$. In practice, the complete feature selection procedure (including the empirical distributions computation) can rank a set of $10^5$ features in about 30 seconds on a standard PC.

## 7.3 Fitting PH distributions

Once the features have been selected, the associated FPT are modeled with general phase-type (PH) distributions. A PH distribution is defined as the

distribution of the time to absorption in an absorbing MC (see section 1.4). These distributions are used here as they can conveniently model discrete time durations. For instance, they are used in many stochastic models such as queuing systems in order to model service times [80]. This section is concerned with the estimation of PH distributions from the observed FPT. Subsection 7.3.1 presents the PHit algorithm while subsection 7.3.2 illustrates the influence of the number of phases used to fit the data.

### 7.3.1 PHit algorithm

In this section, we introduce the PHit algorithm for fitting discrete PH distributions. This algorithm is used to fit the FPT associated to the selected features $\mathcal{P}$ with PH distributions (see step 3 in Figure 7.1). Here, the sample does not directly correspond to the learning sequences of the classification task but to FPT extracted from these sequences. PHit can be thought of as a simplification of the POMMPHit algorithm (see section 5.5.2) which fits several PH distributions with a POMM while PHit aims at fitting a single PH distribution with an absorbing MC. These algorithms mainly differ in the definitions of the forward and backward variables and in the maximization step of the EM algorithm.

The EMpht algorithm, developed by Asmussen and Olsson [5], fits continuous PH distributions. In contrast, we focus here on discrete PH distributions. In particular, PHit deals with negative binomial state duration distribution in an absorbing MC while these durations are modeled in EMpht by a negative exponential density, typical of continuous Markov processes. The re-estimation formula in both algorithms are thus distinct. Bobbio *et al.* [13] also proposed a technique for fitting discrete PH distributions, however it is restricted to a particular class of PH distributions (acyclic PH distributions) while PHit can deal with general PH distributions.

Given a set of observed FPT $Z = \{(z_1, c_1), \ldots, (z_l, c_l)\}$ (see definition 41) and a number $n_s$ of states, PHit estimates the parameters $(\boldsymbol{u}, F)$ of a PH distribution $\varphi$ with $n_s - 1$ phases that maximize the likelihood with respect to $Z$. The complete transition matrix, including $F$, of the absorbing MC associated to $\varphi$ is denoted by $A$ (see equation (1.11) for details on the block-partitioning of $A$). To do so, the iterative Expectation-Maximization (EM) algorithm is used [36]. The basic idea is to consider each $z_i$ as an incomplete observation of the underlying process $\{X_t\}$ (an absorbing MC). More precisely, we observe the time to absorption of a process realization but not the hidden path (*i.e.* the sequence of states) reached by the process during this realization. Let $\mathcal{H} = \{h_1, \ldots, h_l\}$ be the set of hidden sequences paths associated to $Z$, where $h_i$ is a sequence of $z_i$ states. We introduce

below two kinds of auxiliary variables which provide sufficient statistics to compute the complete likelihood function:

- $S^{\varphi}(q)$: the number of observations in $\mathcal{H}$ starting in state $q$.
- $N^{\varphi}(q, q')$: the number of times state $q'$ immediately follows state $q$ in $\mathcal{H}$.

The likelihood of the complete observations with respect to the PH distribution $\varphi$:

$$P[Z, \mathcal{H} \mid \varphi] = \prod_{q \in Q^T} u_q^{S^{\varphi}(q)} \, e_q^{N^{\varphi}(q, q_{n_s})} \prod_{q' \in Q_T} F_{qq'}^{N^{\varphi}(q, q')} \tag{7.4}$$

where $q_{n_s}$ denotes the unique absorbing state of the MC associated to $\varphi$ and $\boldsymbol{e}$ is the absorption vector of this chain which can be computed from the transition matrix $F$ between transient states as $\boldsymbol{e} = (I - F)\mathbf{1}$.

The EM algorithm finds the maximum likelihood estimates of the parameters $\rho = (\boldsymbol{u}, F)$ of the joint distribution $P[Z, \mathcal{H} \mid \rho]$ where the variable $\mathcal{H}$ is unobserved ($\mathcal{H}$ is a *latent* or *hidden* variable). This is achieved by computing iteratively the parameters $\rho$ that maximize $\mathbb{E}\left[\ln(P[Z, \mathcal{H} \mid \rho]) \mid Z\right]$. Each EM iteration involves two steps: (i) the computation of the conditional expectation of the latent variables given the last parameter values $\rho^{t-1}$ and the partial observations $Z$, (ii) the maximization of the parameters $\rho^t$ given the conditional expectation of the latent variables. The computations of these two steps in the PHit algorithm are detailed hereafter.

### Expectation step

In PHit, the latent variables are the auxiliary variables $S^{\varphi}(q)$ and $N^{\varphi}(q, q')$. Their conditional expectations are conveniently obtained using *forward* and *backward* variables defined in a similar way as in the POMMPHit algorithm (see section 5.5.2). Given a state $q \in Q$ and a time $t \in \mathbb{N}$, the forward variable $\alpha^{\varphi}(q, t)$ computes the probability of being in state $q$ after $t$ steps while starting in a transient state.

$$\alpha^{\varphi}(q, t) = P[X_t = q, \{X_k\}_{k=1}^{t-1} \in Q_T \mid X_0 \in Q_T] \tag{7.5}$$

The forward variables can be computed using the following recurrence for $q \in Q$ and $t \in \mathbb{N}$:

$$\begin{aligned} \text{(Base case)} \quad \alpha^{\varphi}(q, 0) &= \begin{cases} u_q & \text{if } q \in Q_T \\ 0 & \text{otherwise} \end{cases} \\ \text{(Induction)} \quad \alpha^{\varphi}(q, t) &= \textstyle\sum_{q' \in Q_T} \alpha^{\varphi}(q', t-1) A_{q'q} \end{aligned} \tag{7.6}$$

Given a state $q \in Q$ and a time $t \in \mathbb{N}$, the backward variable $\beta^{\varphi}(q,t)$ computes the probability that state $q$ is reached by the process $t$ steps before getting absorbed:

$$\beta^{\varphi}(q,t) = P[X_0 = q, \{X_k\}_{k=1}^{t-1} \in Q_T \mid X_t = q_{n_s}] \tag{7.7}$$

The following recurrence computes the backward variables for $q \in Q$ and $t \in \mathbb{N}$:

$$\begin{aligned} \text{(Base case)} \quad \beta^{\varphi}(q,0) &= \begin{cases} 1 & \text{if } q = q_{n_s} \\ 0 & \text{otherwise} \end{cases} \\ \text{(Induction)} \quad \beta^{\varphi}(q,t) &= \begin{cases} 0 & \text{if } q = q_{n_s} \\ \sum_{q' \in Q} \beta^{\varphi}(q', t-1) A_{qq'} & \text{otherwise} \end{cases} \end{aligned} \tag{7.8}$$

The probability $\varphi(k)$ to get absorbed in $k$ steps (see equation (1.40)) can be computed as follows using the forward or backward variables:

$$\varphi(k) = \alpha^{\varphi}(q_{n_s}, k) = \sum_{q_0 \in Q_T} u_{q_0} \beta^{\varphi}(q_0, k) \tag{7.9}$$

The forward variables $\alpha_j^{\varphi}(t)$ can be thought of as a simplification of the forward variables used in the Baum-Welch algorithm (see section 2.3.3) which estimates the parameters of Hidden Markov Models (HMMs). Indeed, in the Baum-Welch algorithm, a forward variable computes the probability of being in a state after having generated a given string. In PHIT, a forward variable computes the probability of being in a given state after $t$ steps no matter how this state is reached.

**Conditional expectation of $S^{\varphi}(q)$:** The variables $S^{\varphi}(q)$ where $q \in Q_T$ can be decomposed as a sum of indicator variables $S^{\varphi}(q) = \sum_{k=1}^{l} \mathbb{1}\{h_{k,0} = q\}$ where the notation $h_{k,j}$ stands for the $j$-th state in the $k$-th hidden path. The conditional expectation $\overline{S^{\varphi}}(q) = \mathbb{E}\left[S^{\varphi}(q) \mid (z_1, c_1), \ldots, (z_l, c_l)\right]$ is given by

$$\begin{aligned} \overline{S^{\varphi}}(q) &= \mathbb{E}\left[\sum_{k=1}^{l} \mathbb{1}\{h_{k,0} = q\} \,\middle|\, (z_1, c_1), \ldots, (z_l, c_l)\right] \\ &= \sum_{k=1}^{l} c_k \mathbb{E}\left[\mathbb{1}\{h_{k,0} = q\} \mid z_k\right] \\ &= \sum_{k=1}^{l} c_k P[h_{k,0} = q \mid z_k] \end{aligned} \tag{7.10}$$

The conditional probability $P[h_{k,0} = q \mid z_k] = \frac{P[h_{k,0}=q, z_k]}{P[z_k]}$ with $P[h_{k,0} = q, z_k] = u_q \beta^{\varphi}(q, z_k)$ and $P[z_k] = \alpha^{\varphi}(q_{n_s}, z_k)$. Finally, the conditional expec-

tation is defined as

$$\overline{S^{\varphi}}(q) = \sum_{k=1}^{l} c_k \frac{u_q \beta^{\varphi}(q, z_k)}{\alpha^{\varphi}(q_{n_s}, z_k)} \tag{7.11}$$

**Conditional expectation of $N^{\varphi}(q, q')$:** The variables $N^{\varphi}(q, q')$ can be decomposed as $\sum_{k=1}^{l} \sum_{t=0}^{z_k - 1} \mathbb{1}\{h_{k,t} = q, h_{k,t+1} = q'\}$, that is, counting the presence of the transition $q \to q'$ in each position of each hidden path. The conditional expectation $\overline{N^{\varphi}}(q, q') = \mathbb{E}\left[N^{\varphi}(q, q') \mid (z_1, c_1), \ldots, (z_l, c_l)\right]$ is obtained as follows:

$$\begin{aligned} \overline{N^{\varphi}}(q, q') &= \mathbb{E}\left[\sum_{k=1}^{l} \sum_{t=0}^{z_k - 1} \mathbb{1}\{h_{k,t} = q, h_{k,t+1} = q'\} \,\middle|\, (z_1, c_1), \ldots, (z_l, c_l)\right] \\ &= \sum_{k=1}^{l} c_k \sum_{t=0}^{z_k - 1} P[h_{k,t} = q, h_{k,t+1} = q' \mid z_k] \end{aligned} \tag{7.12}$$

The joint probability $P[h_{k,t} = q, h_{k,t+1} = q', z_k] = \alpha^{\varphi}(q, t) A_{qq'} \beta^{\varphi}(q', z_k - t - 1)$, that is the probability of starting in any transient state, to reach state $q$ after $t$ steps, then to perform the transition $q \to q'$, and finally to get absorbed $z_k - t - 1$ steps later. The expectation $\overline{N^{\varphi}}(q, q')$ is finally provided by

$$\overline{N^{\varphi}}(q, q') = \sum_{k=1}^{l} c_k \sum_{t=0}^{z_k - 1} \frac{\alpha^{\varphi}(q, t) A_{qq'} \beta^{\varphi}(q', z_k - t - 1)}{\alpha^{\varphi}(q_{n_s}, z_k)} \tag{7.13}$$

### Maximization step

Once the expected values of the latent variables have been computed, the maximum likelihood estimators of the PH distribution parameters are the following:

$$\begin{aligned} u_q &= \frac{\overline{S^{\varphi}}(q)}{\sum_{q' \in Q_T} \overline{S^{\varphi}}(q')} \quad \forall q \in Q_T \\ F_{qq'} &= \frac{\overline{N^{\varphi}}(q, q')}{\sum_{q' \in Q} \overline{N^{\varphi}}(q, q')} \quad \forall q, q' \in Q_T \end{aligned} \tag{7.14}$$

The parameters $(\boldsymbol{u}, F)$ are initialized at random at the beginning of the algorithm. PHit iterates until the relative likelihood improvement falls below a user-defined threshold. In our experiments, PHit is rather insensitive to the initialization. In other words, the value of the likelihood obtained after convergence is rarely affected by different initializations. However, the parameters found by PHit can be different using different initializations since the representation of a PH distribution is not unique. As in the

POMMPHIT algorithm, the forward and backward variables can efficiently be computed using a lattice data structure. The computational complexity per iteration is $\Theta(n_t L^2)$ where $L$ is the longest observed FPT and $n_t$) is the number of transitions in the absorbing MC. An alternate upper bound, which is tight if the absorbing MC has a dense transition graph, is $\mathcal{O}(L^2 n_s^2)$. PHIT has been implemented in the ANSI C language and it can estimate PH distributions with 100 phases over 2,000 events in about one minute on a standard PC.

### 7.3.2 Influence of the number of phases

The PHIT algorithm estimates the parameters of a general PH distribution $\varphi$ having a prescribed number $n_s - 1$ of phases. This is a structural parameter as it defines the number of transient states in the absorbing MC associated to $\varphi$. Importantly, the number of phases defines a trade-off between the simplicity of the model and the accuracy of the fit. In other words, this parameter controls the overgeneralization-overfitting or *bias-variance* trade-off. Modeling FPT with a small number of phases gives a larger probability mass to infrequently or not observed FPT. In this sense, a small number of phases bias towards generalization. Therefore, PH modeling can be thought of as a technique for smoothing empirical FPT distributions. In opposite, modeling FPT with a large number of phases tends to overfit the empirical FPT distribution.

To illustrate the influence of the number of phases, let us considering the following example. We have artificially generated a set of FPT defined as follows $Z = \{(1,7),(2,2),(3,4),(4,2),(5,3),(6,5),\ (7,2),(9,2),(10,1)\}$. Note that there is no observed FPT equal to 8. Then, we apply PHIT on this sample for increasing numbers of phases. Figure 7.2 shows the PH distributions obtained while fitting $Z$ with 1, 3, 5 and 10 phase(s). The dashed line denotes the empirical FPT distribution and the solid line stands for the PH distribution obtained with PHIT. Using a single phase roughly models the observations as the tail of the PH distribution does not fit the modes (*i.e.* the local maxima) present after time 1. Note that the obtained PH distribution gives a strictly positive probability to the unseen FPT 8. Increasing the number of phases to 5 fits more closely the empirical FPT distribution as the 3 first modes of the PH distribution approximately coincide with those observed in $Z$. Note also that the FPT equal to 8 has still a strictly positive probability. Using 10 phases, the PH distribution nearly reproduces the empirical distribution and the FPT 8 has here a null probability to occur. More generally, we observe that using a larger number of phases better fits the tail of the distribution. From the smoothing point of view, larger FPT

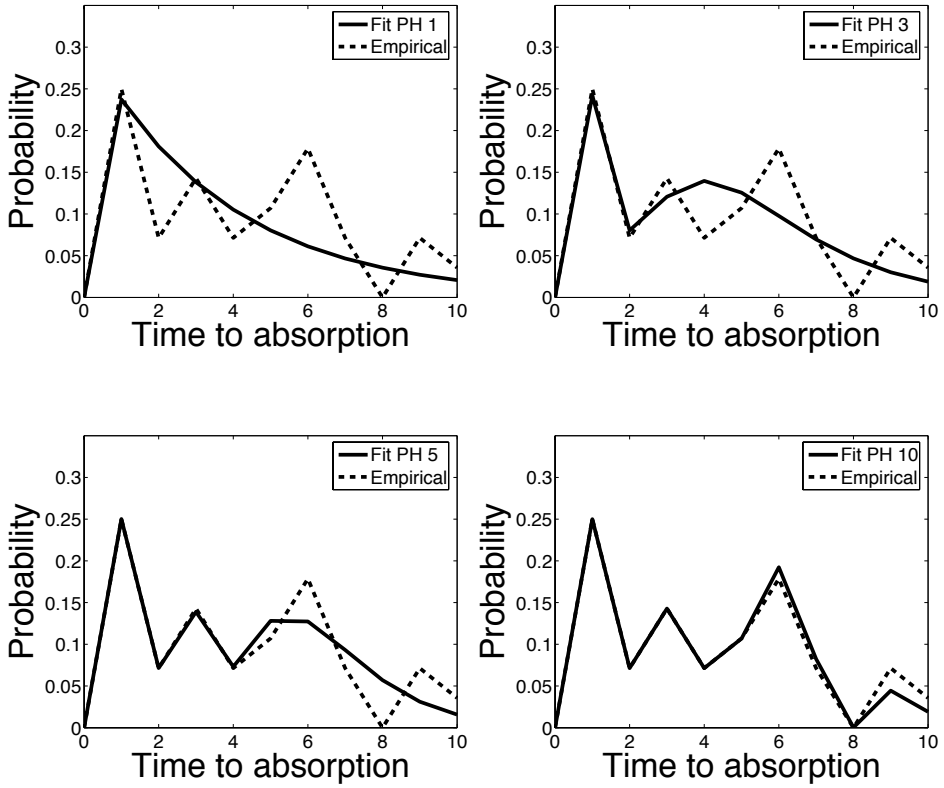



Figure 7.2: The influence of the number of phases while fitting the FPT $Z = \{(1,7),(2,2),(3,4),(4,2),(5,3),(6,5),(7,2),(9,2),(10,1)\}$ with increasing numbers of phases. The dashed line denotes the empirical FPT distribution while the solid line represents the PH distribution obtained with PHit.

are smoothed first while decreasing the number of phases. This is consistent with the implicit assumption that the observed FPT are drawn from a PH distribution. Indeed, longer absorption times occur generally less frequently in an absorbing MC and therefore statistics associated with these times are less reliable and need to be smoothed first. In our sequence classification context, the number of phases has to be selected to provide the best classification accuracy. To do so, the $n_s$ parameter is tuned on an independent validation set of sequences (see section 7.5.2).

## 7.4 Building PH-based classifiers

This section presents two classifiers based on the PH distributions estimated by the PHit algorithm detailed in section 7.3. Let us recall that the FPT are fitted separately for each class $y \in \mathcal{Y}$. The PH distribution associated to a feature $(v^1, v^2)$ estimated from the sequences belonging to a class $y \in \mathcal{Y}$ is denoted by $\varphi^y_{(v^1,v^2)}(.)$. The first classifier consists of a MAP decision rule where the sequence likelihood is computed using PH distributions. The second classifier is a SVM applied in a feature space constructed from PH distributions. These two classifiers are respectively presented in subsections 7.4.1 and 7.4.2.

### 7.4.1 Maximum a posteriori classifier

In this section, we introduce a maximum *a posteriori* (MAP) classifier based on PH distributions. For each class $y \in \mathcal{Y}$, the set of PH distributions $\{\varphi^y_{(v^1,v^2)}(.)\}_{(v^1,v^2)\in\mathcal{P}}$ defines a generative model of the FPT in the class $y$. It is assumed here that the features are independent. That is, the FPT associated to a pair $(v^1, v^2) \in \mathcal{P}$ are independent from the FPT associated to any other pair $(v^3, v^4) \in \mathcal{P}$. Consequently, the log-likelihood of sequence $s$ with respect to the model $\{\varphi^y_{(v^1,v^2)}(.)\}_{(v^1,v^2)\in\mathcal{P}}$ is defined as

$$\begin{aligned} \log(P[s \mid y]) &= \sum_{(v^1,v^2)\in\mathcal{P}} \log(P[\mathrm{FPT}_s(v^1, v^2) \mid y]) \\ &= \sum_{(v^1,v^2)\in\mathcal{P}} \sum_{(z,c)\in\mathrm{FPT}_s(v^1,v^2)} c \ \log(\varphi^y_{(v^1,v^2)}(z)) \end{aligned} \tag{7.15}$$

As usual for models making this naive assumption, the independence is not always satisfied but good results are obtained in practice. Predicting the label of a sequence $s$ is made by selecting the class that maximizes the posterior log-probability:

$$y^* = \arg\max_y \left(\log(P[s \mid y]) + \log(P[y])\right) \tag{7.16}$$

where $P[y]$ denotes the prior probability of class $y$ which is estimated as the proportion of sequences belonging to $y$ in the complete training set.

### 7.4.2 Phase-type kernel

We introduce here the PH kernel which maps the sequences in a feature space based on PH distributions. There are several ways to incorporate generative information in the design of a kernel function. We use here a technique called *marginalization kernels* [118]. First, a set $\mathcal{M}$ of generative models and a prior probability over these models $P[M]$ with $M \in \mathcal{M}$ are defined. We consider the probability $P[\boldsymbol{x}, \boldsymbol{z}, M]$ of observing jointly a model $M$ and two instances $\boldsymbol{x}$ and $\boldsymbol{z}$. The probability $P[\boldsymbol{x}, \boldsymbol{z}]$ of observing jointly $\boldsymbol{x}$ and $\boldsymbol{z}$ is obtained by *marginalizing*, *i.e.* summing over all models $M \in \mathcal{M}$, the distribution $P[\boldsymbol{x}, \boldsymbol{z}, M]$:

$$P[\boldsymbol{x}, \boldsymbol{z}] = \sum_{M \in \mathcal{M}} P[\boldsymbol{x}, \boldsymbol{z} \mid M]\ P[M] \tag{7.17}$$

A marginalization kernel computes precisely this probability while making the assumption that $\boldsymbol{x}$ and $\boldsymbol{z}$ are conditionally independent given $M$:

$$k(\boldsymbol{x}, \boldsymbol{z}) = P[\boldsymbol{x}, \boldsymbol{z}] = \sum_{M \in \mathcal{M}} P[\boldsymbol{x} \mid M]\ P[\boldsymbol{z} \mid M]\ P[M] \tag{7.18}$$

The marginalization kernel $k(.,.)$ is positive definite (see theorem 5) as it corresponds to the dot product in the feature space defined by the following mapping:

$$\Phi : \boldsymbol{x} \rightarrow \{P[\boldsymbol{x} \mid M]\sqrt{P[M]}\}_{M \in \mathcal{M}} \tag{7.19}$$

That is, the feature space is indexed by the generative models $M \in \mathcal{M}$. The value of an instance $\boldsymbol{x}$ along the axis corresponding to the model $M$ is the likelihood $P[\boldsymbol{x}|M]$ of $\boldsymbol{x}$ with respect to $M$ weighted by the square root of the prior probability of $M$. It is sometimes important to replace the involved probabilities by log-probabilities in order to avoid numerical issues. The resulting kernel computes therefore a dot product between vectors made of log-probabilities. Strictly speaking, the kernel no longer corresponds to a joint (log-)probability but this is not required to perform classification with a SVM.

We now detail how this technique is applied to build a kernel from PH distributions. For each feature $(v^1, v^2)$, a marginalization kernel $k_{(v^1,v^2)}(.,.)$ computing the probability that two sequences $s$ and $s'$ have been generated together is introduced. The collection of generative models is here defined as the PH distributions relative to the FPT from $v^1$ to $v^2$ estimated separately

for each class label: $\mathcal{M} = \{\varphi^y_{(v^1,v^2)}(.)\}_{y\in\mathcal{Y}}$. These generative models can therefore be indexed by the set of labels $\mathcal{Y}$. The marginalization kernel with respect to the feature $(v^1, v^2)$ is defined as follows:

$$\begin{aligned} k_{(v^1,v^2)}(s,s') &= P[s,s'] = \sum_{M\in\mathcal{M}} P[s \mid M]\ P[s' \mid M]\ P[M] \\ &= \sum_{y\in\mathcal{Y}} P[\mathrm{FPT}_s(v^1,v^2) \mid y]\ P[\mathrm{FPT}_{s'}(v^1,v^2) \mid y]\ P[y] \end{aligned} \tag{7.20}$$

where

$$P[\mathrm{FPT}_s(v^1,v^2) \mid y] = \prod_{(z,c)\in \mathrm{FPT}_s(v^1,v^2)} [\varphi^y_{(v^1,v^2)}(z)]^c \tag{7.21}$$

and where $P[y]$ denotes the prior probability of the class $y$ which is estimated as the proportion of sequences belonging to $y$ in the complete training set. The mapping induced by this kernel is

$$\Phi^{(v^1,v^2)} : s \rightarrow \left(\sqrt{P[y]}\ P[\mathrm{FPT}_s(v^1,v^2) \mid y]\right)_{y\in\mathcal{Y}} \tag{7.22}$$

The PH kernel is obtained by summing the marginalization kernels over all the features $(v^1, v^2) \in \mathcal{P}$:

$$k^{\varphi}(s,s') = \sum_{(v^1,v^2)\in\mathcal{P}} k_{(v^1,v^2)}(s,s') \tag{7.23}$$

The PH kernel is positive definite as a sum of positive definite functions is also positive definite. Furthermore, the PH kernel amounts to compute the dot product in a feature space defined by the following mapping:

$$\Phi^{\mathcal{P}} : s \rightarrow \left(\sqrt{P[y]}\ P[\mathrm{FPT}_s(v^1,v^2) \mid y]\right)_{\substack{y\in\mathcal{Y} \\ (v^1,v^2)\in\mathcal{P}}} \tag{7.24}$$

The SVM classifier based on the PH kernel defines a hybrid approach coupling the generative part of PH distributions with the discriminative part of SVM. The statistical learning theory of Vapnik (see section 3.1), providing theoretical foundations to SVM, assumes that the training instances are generated i.i.d. from some source distribution. In our case, the feature space is built from all the training sequences. Therefore, the training instances are no longer independent in the feature space. To face this issue, the PH distributions used to compute the PH distributions $\varphi^y_{(v^1,v^2)}(.)$ are estimated from a part of the training data (80 % in our experiments) and the rest of the data is used to train the SVM. Once the SVM has been trained, new sequences are classified by looking at which side of the hyperplane they lie in the PH feature space (see the SVM decision function defined in equation (3.25)).

## 7.5 Experiments

The preceding section has presented two classifiers based on PH distributions: (i) a MAP classifier computing the sequence likelihood using PH distributions and (ii) a SVM along with a kernel mapping the sequences in a feature space built from PH distributions. In this section, the performances of both classifiers are assessed on two sequence classification tasks: (i) DNA splicing region detection (`Splice` dataset) and (ii) protein sublocalization (`DBSubloc` database). First, the influence of the parameters, *i.e.* the number of phases and the number of features, is studied. Then, comparative results, in terms of the classification accuracy, are provided versus a smoothed $N$-gram classifier (see section 6.1.1) and a SVM based on the blended spectrum kernel (see section 6.2.2).

### 7.5.1 Dataset descriptions

This subsection details the `Splice` and `DBSubloc` datasets as well as the classification tasks considered in our experiments.

#### Splice

The `Splice` dataset is publicly available from the UCI Machine Learning Repository. Splice junctions are points on a DNA sequence at which 'superfluous' DNA is removed during the process of protein creation in higher organisms. The problem posed in this dataset is to recognize, given a sequence of DNA, the boundaries between exons (the parts of the DNA sequence retained after splicing) and introns (the parts of the DNA sequence that are spliced out). This problem consists of two subtasks: recognizing exon/intron boundaries (referred to as EI sites), and recognizing intron/exon boundaries (IE sites). In the biological community, IE borders are referred to as *acceptors* while EI borders are referred to as *donors*. More specifically, given a position in the middle of a window of 60 DNA sequence elements (called "nucleotides" or "base-pairs"), one has to decide whether this is an "intron ⇒ exon" boundary (IE), an "exon ⇒ intron" boundary (EI), or neither of them (N). We restrict here our attention to binary classification problems using only sequences labeled either as EI or IE. Apart from its class label, each instance is characterized by 60 DNA nucleotides found at a given position in the window. The alphabet is composed of 8 letters[3]. The class priors are equal and the training, validation and test sets contain respectively 975, 253 and 253 sequences.

[3] `A, T, C, G` stands for the specific nucleotides and 4 additional characters are used to indicate ambiguity: `D={A,G,T}; N={A,G,C,T};S={C,G}; R={A,G}`.

### DBSubloc

The DBSubloc database[4] contains protein sequences (primary structures) with their subcellular localization for various organisms such as eukaryotes, bacteria, plants or animals, to name a few. In general, the protein sublocalization task consists in predicting the localization of a protein in a cell given its primary structure. This task can be thought of as a sequence classification problem where the classes are the possible localizations in the cell, *e.g.* the mitochondria, the ribosomes or the membrane. The classification task considered here consists in finding if a protein from a plant organism is located in the membrane or in the mitochondria of the cell. In contrast with the sequences from the Splice dataset, the protein sequences considered here have different lengths. The average length of the sequences is 406. The class priors are respectively 0.45 and 0.55 for the membrane and the mitochondria classes and the training, validation and test sets contain respectively 151, 51 and 50 sequences.

## 7.5.2 Influence of parameters

The influence of parameters (the number of phases and the number of features) is evaluated with both classification methods using the Splice validation data. For computational reasons, the influence of the maximal subsequence length $N$ has not been systematically checked. In all our experiments this parameter has been set to $N = 3$ and therefore all the features made of subsequences up to length 3 are considered (what subsumes $N = 1$ and $N = 2$). From a practical point of view, using larger $N$ values makes the feature selection computationally very intensive, especially in the case of the DBSubloc database. The learning curves presented hereafter are produced by extracting samples of growing sizes from the training set. For each data size (except for 100%), 10 samples have been randomly extracted from the training set in order to produce averaged results with standard deviations. The SVM classifiers are trained using SVM$^{light}$ 6.01 [69]. The regularization constant $C$ is tuned according to the heuristic of Joachims:

$$C = \frac{1}{\frac{1}{n}\sum_{i=1}^{n}\sqrt{k(s^i, s^i)}} \tag{7.25}$$

As mentioned in section 7.4.2, in the case of the PH kernel, 80 % of the training data are used to estimate the PH distributions and the remaining 20% are used to train the SVM classifier.

[4] DBSubloc is available at http://www.bioinfo.tsinghua.edu.cn/dbsubloc.html.

### Influence of the number of phases

The top part of Figure 7.3 shows the accuracy obtained on validation data using the MAP classifier with an increasing number of phases and 100 features. Interestingly, one can observe that for very small training set sizes (2% and 5%), using 2 phases leads to significantly better results as for 2% of the training data, the classification accuracy is about 75% whereas it is approximatively 56% when using 5 or 10 phases. The reason is that the estimation of PH distributions with a larger number of parameters[5] becomes unreliable when there are too few observations. Furthermore, we argue in section 7.3.2 that decreasing the number of phases acts as a smoothing procedure for the empirical FPT distribution, which is especially important when only a few data is available. When the training set size becomes greater than 10%, the benefit of additional phases is noticed. The experiments were made using up to 20 phases but it appears that using more than 5 phases does not significantly improve the accuracy for this problem. It should be pointed out that an accuracy close to 85% was eventually attained using 2 phases while this score was already obtained with 50% of the training data using 5 or 10 phases.

The bottom part of Figure 7.3 shows the influence of the number of phases on the SVM classifier based on the PH kernel, when using 100 features. One can observe that this parameter has nearly no influence on the results. The SVM classifier thus seems able to compensate unreliable estimations (obtained when estimating a PH distribution containing many phases from a small amount of data) by adapting its decision function.

### Influence of the number of features

The top part of Figure 7.4 shows the accuracy obtained on validation data using the MAP classifier with an increasing number of features (1, 4, 30, 100 and 1000) and 5 phases. The features considered in our experiments are made of subsequences up to length $N = 3$. Features have been selected according to their JS ranking (see section 7.2). It can be observed that, in general, the classification accuracy increases with the number of selected features. An accuracy around 73 % is already obtained using a single feature (the highest ranked feature is the subsequence pair $(\texttt{T}, \texttt{GG})$). Using 1000 features only improves the results when 50% of the training data are used as for smaller training set sizes, the lack of observations (for rare features) decreases the accuracy of the fitting. The influence of the number of features on the SVM based classifier is shown in the bottom part of Figure 7.4. Using more features clearly improves the classification accuracy obtained on

[5] A PH distribution with $p$ phases has $p^2 + p$ parameters.

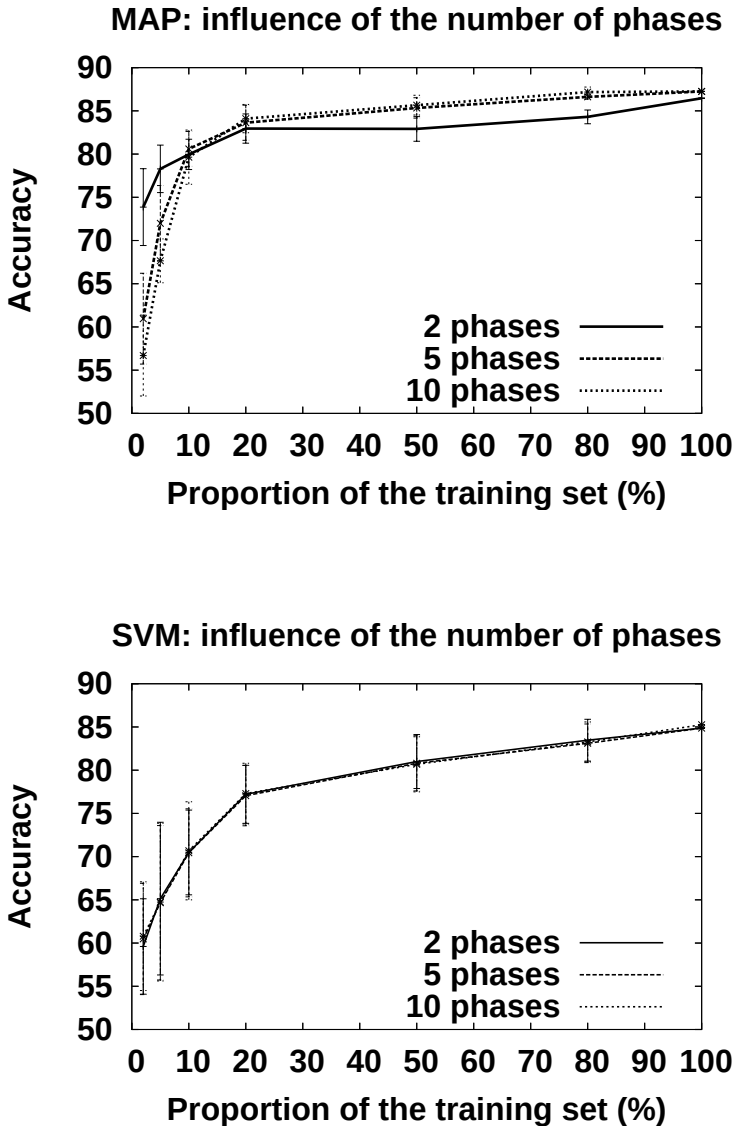



Figure 7.3: Influence of the number of phases on both classifiers using 100 features, evaluated on the `Splice` dataset. Top: influence of the number of phases on the MAP classifier. Bottom: influence of the number of phases on the SVM classifier along with the PH kernel.

the validation data. In contrast with the MAP classifier, this is also true when using 1000 features and when only a small amount of training data is available. Again, we can conclude that SVM is not much hindered by poor PH estimations as the maximization of the margin makes the classifier robust to noisy data.

### 7.5.3 Comparative results

Figure 7.5 shows comparative results on both dataset using several classifiers: (i) smoothed N-grams (see section 6.1.1), (ii) the MAP classifier based on PH distributions, (iii) SVM with the PH kernel and (iv) SVM with the blended[6] spectrum kernel (see section 6.2.2). All the hyperparameters, except the regularization constant $C$ (see equation (7.25)) are tuned on the validation set. The top part of Figure 7.5 presents results obtained on the `Splice dataset`. The best parameters obtained on validation data are a 4-gram, a blended string kernel length of 7 with a length weight of 3.5, and a 10-phase PH modeling of the 1000 best features for our methods. When a sufficient amount of data is used (at least 50% of the training data), the best performances are obtained with the MAP classifier. However, for smaller amounts of data (up to 20% of the training set), the SVM classifiers obtain better performances than the MAP classifier. Both kernels have comparable performances on this task. The test classification accuracy for the 4-grams, the PH kernel, the blended spectrum string kernel, and the MAP classifiers are respectively 84.1%, 88.5%, 88.8% and 89.7% when the whole training set is used.

The bottom of Figure 7.5 presents results obtained on the `DBSubloc` database. The best parameters obtained on validation data are a 2-gram, a blended spectrum string kernel length of 4 with a length weight of 2.5 and a 5-phase PH modeling of the 100 best features for our methods. The test classification accuracy for 2-grams, the blended spectrum kernel, the MAP classifier and the PH kernel are respectively 80.4%, 82.35%, 82.4% and 84.3% when the whole training set is used.

### 7.5.4 Implementation issues

The implementation of the proposed classification techniques consists of 4 modules (i) the feature selection procedure, (ii) the PHit algorithm, (iii) the MAP classifier and (iv) the PH kernel precomputation. All these modules have been implemented in the ANSI C programming language for efficiency purpose. The PHit algorithm has been first developed in the Matlab 7.1

[6] Experiments with the standard $p$-spectrum kernel offer worse results than the blended version reported here.

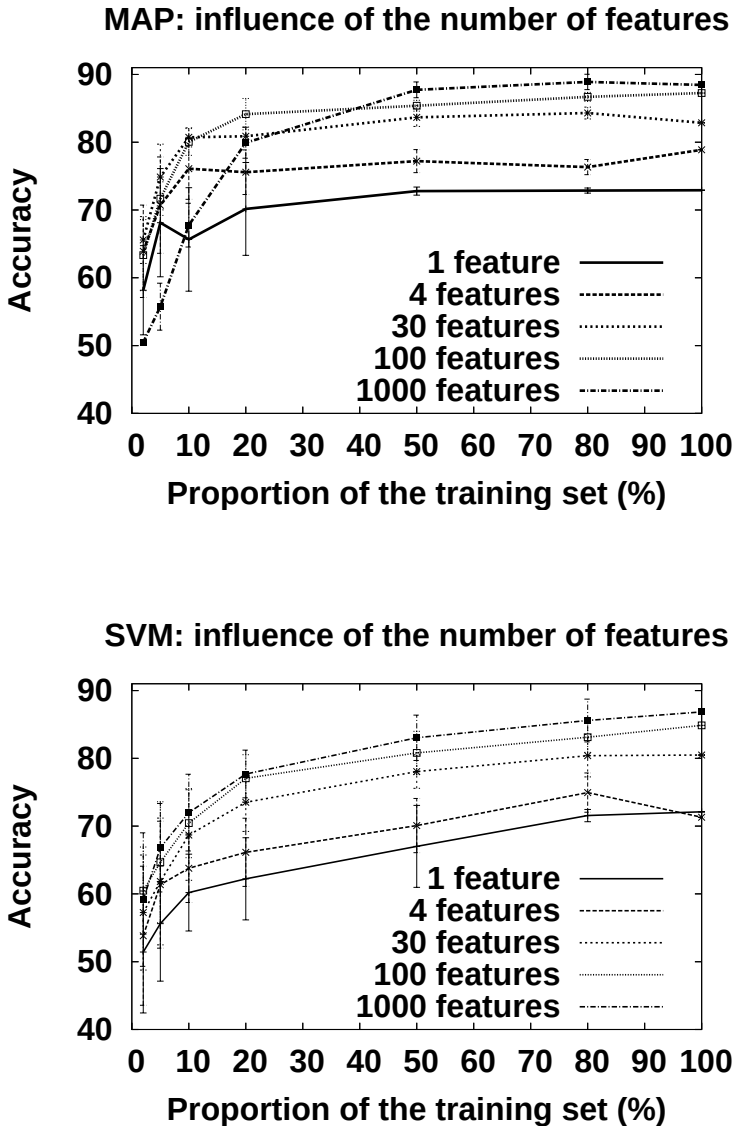



Figure 7.4: Influence of the number of features on both classifiers using 5 phases, evaluated on the `Splice` dataset. Top: the influence of the number of features on the MAP classifier. Bottom: influence of the number of features on the SVM classifier along with the PH kernel.

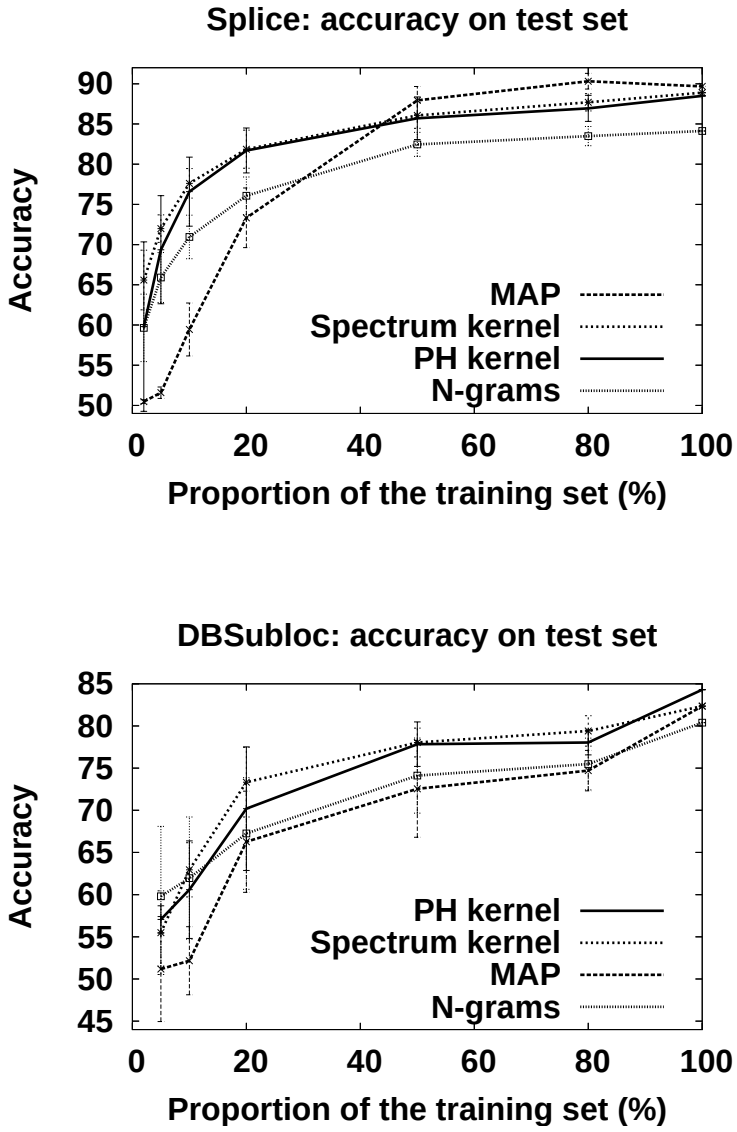



Figure 7.5: Comparative classification results for N-grams, MAP and SVM with the PH kernel and the spectrum kernel on `Splice` (top) and `DBSubloc` (bottom) test data. The hyperparameters of each method have been tuned on validation data.

programming language. This first prototype could only handle PH distributions with a few phases (about 10 phases) due to the slow execution of programs containing nested loops in Matlab. In contrast, the ANSI C prototype is much more efficient and can deal with PH distributions having about 10 times more phases. It should be noted that the PH kernel precomputation module does not actually compute a Gram or a kernel matrix. Instead, it explicitly computes the representation of the sequences in the feature space (see equation (7.24)). These representations are fed to SVM$^{light}$ 6.01 [69] which is trained using a linear kernel (corresponding to the PH kernel). Classification of new sequences is performed applying the same procedure.

## 7.6 Summary and discussion

We have proposed a novel approach to the classification of discrete sequences. It relies on First Passage Times (FPT) features which are the central theme of this thesis. The use of such features brings a novel characterization of sequences as one focuses on *when* events of interest occur rather than *which* events occur frequently. Our approach attempts to exploit the temporal dynamics in sequences to perform sequence classification. More precisely, it relies on the following assumption: sequences belonging to distinct classes are drawn from generative models having different FPT dynamics. Consequently, the FPT are considered here as discriminative informations used to perform sequence classification. Furthermore, we argue that FPT are global features of the sequences to be classified as there is no *a priori* fixed maximum time between two events. Therefore, such features are well-suited to capture long-range dependencies or complex time-dependent dynamics in sequences.

We have proposed a general learning scheme to perform sequence classification using the FPT. It consists of 3 stages: (i) feature extraction and selection (ii) fitting the FPT with phase-type distributions and (iii) building a classifier based on the estimated PH distributions. Since the number of features can be potentially large, a feature selection procedure is proposed to keep a limited number of informative features. To do so, features are ranked according to the Jensen-Shannon (JS) divergence to their class conditional *empirical* FPT distributions. The most diverging features are kept for the classification process. The FPT associated to the selected features are modeled with general phase-type (PH) distributions. These distributions are very powerful since they can decompose complex distributions into a combination of phases. Furthermore, the FPT dynamics in rich models such as POMMs or HMMs has been shown to be of phase-type (see sec-

tion 5.3). Therefore, using PH distributions to model FPT offers a way to incorporate a part of the expressiveness of HMMs into sequence classifiers. The PHit algorithm, an adapted version of the Expectation-Maximization algorithm, is proposed to estimate PH distributions from data. It can be thought of as a simpler version of the POMMPHIT algorithm used to estimate the probabilistic parameters of a POMM so as to fit the observed FPT. In contrast with this technique, PHIT aims at fitting the FPT associated to a single feature $(v^1, v^2)$ with an absorbing MC. Tuning the number of phases deals with the *bias-variance* trade-off since using a small number of phases spreads the probability mass to infrequently or unobserved passage times while using many phases tends to overfit the observed FPT. The selected features are used in two classification schemes: a maximum *a posteriori* (MAP) classifier and Support Vector Machines (SVM) classifier along with a new kernel called the *PH kernel*. The MAP classifier uses PH distributions as a generative model to compute the sequence likelihood while the PH kernel computes the dot product in a feature space constructed from PH distributions.

We have presented experimental results obtained by our techniques applied to DNA splicing site detection and on protein sublocalization. First, the influence of the number of phases and of features has been studied. We have shown that increasing the number of phases yields slightly better results, provided enough data is available. Indeed, the estimation of PH distributions with a large number of phases becomes unreliable when there are too few observations. We have also observed that the SVM classifier seems to be able to compensate inaccurate PH estimations by adapting its decision function. We believe that this robustness follows from the maximum margin principle. Then, we have studied the influence of the number of FPT features. These features are ranked using a score based on the JS divergence. As expected, increasing the number of features improves the classification accuracy of the classifiers. However, good results are already obtained when using a few features, *e.g.* 30. This property underlines the efficiency of the proposed feature selection method. Finally, we have presented comparative results with respect to two competing techniques: (i) a smoothed $N$-gram classifier and (ii) SVM based on the blended spectrum kernel. Our methods outperform the $N$-gram classifier on both classification tasks. The MAP classifier provides the best overall result on the splice site detection tasks while the PH kernel and the blended spectrum have a comparable performance. On the protein sublocalization task, the PH kernel shows the best overall results while the MAP classifier is slightly outperformed by the blended spectrum kernel.

We now discuss possible extensions to be addressed in a future research. First, the number of phases could be tuned separately for each considered feature. Indeed, the complexity of the FPT dynamics may vary from one feature to another and the number of phases should be adapted accordingly. A more powerful approach would automatically discover the optimal structure, including the number of phases, of the absorbing MC associated to PH distributions. To do so, we propose to simplify the POMMStruct algorithm (see section 5.5) used to learn the structure of Partially Observable Markov Models (POMMs). Indeed, the structural operations, *i.e.* adding states and trimming transitions, could be readily applied to absorbing MC. Furthermore the estimation procedure PHit is already a simplification of the POMMPHit algorithm used in POMMStruct to estimate the probabilistic parameters of POMMs.

Another possible refinement concerns the feature selection procedure. The current procedure selects the features having a large JS divergence between their class conditional empirical distributions. While each selected feature may have a good discriminative property, there might be some redundancy among the selected features. This is particularly the case, when features have overlapping subsequences, *e.g.* (`aba, baa`) and (`ba, ba`). To face this issue, one can filter out overlapping feature. Additionally, one could couple the JS divergence with a criterion such as the mutual information [57] in order to define a feature score which trades off discriminative efficiency and minimal redundancy.

# Part IV

# Graph mining

# Chapter 8

# Relevant subgraph mining from First Passage Times

*This chapter presents a novel technique to mine relevant subgraphs in large networks. Most of the material presented in this chapter comes from the technical report [43]. The general problem can be stated as follows:* given a weighted graph and a set $\mathcal{K}$ of nodes of interest, one wants to find the subgraph that best explains the relationship between the nodes in $\mathcal{K}$. *For instance, this kind of mining can be applied to analyze the possible influence of genes in the regulation of a given metabolic pathway. This topic is quite different from those presented in the first and second parts of this thesis since the input data are no longer a set of sequences but a graph. However, the First Passage Times (FPT) play a crucial role here as they are used to define* a relevance *criterion for mining subgraphs.*

*The key idea is to interpret the graph as a Markov chain. The states of such a model correspond to the nodes in the original graph and the transition probabilities are defined proportionally to the edge weights. This defines a dynamic view of the graph in which random walks can be performed. A* $\mathcal{K}$-walk *is defined as a random walk starting in a node of interest and ending when any other node of interest is entered for the first time.* $\mathcal{K}$*-walks are therefore directly related to the FPT between nodes of interest. One can compute the expected number of times the walker reaches each node of the chain during* $\mathcal{K}$*-walks. Similar computations can be performed for the edges of the graph. Relevant nodes and edges are then defined as the most frequently used, in average, during* $\mathcal{K}$*-walks. These expectations can be computed in practice using tools from the MC theory presented in chapter 1. Besides, there is a well-known analogy between random walks in MC and the current in an electrical network [40]. In this respect, we provide an electrical interpretation of* $\mathcal{K}$*-walks whenever applied to an undirected input*

*graph.*

*We also propose to consider $\mathcal{K}$-walks of a prescribed length $L$ or up to a prescribed length $L_{max}$. This approach is called* limited $\mathcal{K}$-walks. *Using limited $\mathcal{K}$-walks incorporates a prior knowledge about the maximum path length allowed to explain the relation between nodes of interest. Moreover, tuning the maximum walk length determines the amount of randomness in the walks since short lengths constraint to move only along short paths whereas an infinite maximum length allows one to follow any paths connecting the nodes of interest. An efficient algorithm is proposed to compute node/edge relevances in the case of limited $\mathcal{K}$-walks. It is based on forward and backward recurrences similar to those used in the expectation steps of the* PHit *(see section 7.3.1) and* POMMPHit *(see section 5.5.2) algorithms. When letting $L_{max}$ tend to infinity limited $\mathcal{K}$-walks become equivalent to the standard $\mathcal{K}$-walks. Consequently, they can approximate standard $\mathcal{K}$-walks by considering a large maximum walk length $L_{max}$.*

*Once node and edge relevances have been computed, one has to extract a subgraph modeling the relationships between the nodes of $\mathcal{K}$. Such an extraction trades off the size of the subgraph and its relevance. In other words, maximally relevant subgraphs of minimal size should be returned. In addition, it is often natural to require the extracted graph to be connected. Unfortunately, this defines a combinatorial problem, related to the Steiner tree problem, hard to tackle in practice. To face this issue, we propose several greedy strategies based on a relevance threshold and on a connectedness constraint. Additionally, an* inflation *technique is proposed to increase the relevance contrast to better discriminate relevant and irrelevant parts of the input graph relatively to $\mathcal{K}$.*

*This chapter is organized as follows. Section 8.1 presents an intuitive overview of the proposed approach including running examples. A technical description of the $\mathcal{K}$-walk approach is given in section 8.2. Limited $\mathcal{K}$-walks are detailed in section 8.3. An extension of the proposed approaches to deal with cluster of nodes of interest is presented in section 8.4. Several methods for extracting a subgraph from edge/node relevances are discussed in section 8.5. An inflation technique to increase the relevance contrast is proposed in section 8.6. An electrical analogy as well as a comparison with an existing approach are provided in section 8.7. Finally, section 8.7 shows experimental results obtained with (limited) $\mathcal{K}$-walks applied to the KEGG LIGAND database [55] and to artificially generated graphs.*

## 8.1 Overview and intuition

The general problem is to extract a subgraph that best explains the relationships between $k$ ($\geq 2$) given nodes of interest in a graph. We are considering for instance a large metabolic network which can be represented as a directed connected graph[1] of reactions and bioentities involved in these reactions. More specifically, we could analyze the Methionine Biosynthesis of *Escherichia Coli*. A simplified view of this metabolic pathway[2] is depicted on Figure 8.1. One could study in particular the influence of the *metR Activator* on the production of *Homocysteine* from *L-Homoserine* in this pathway. In this case, there are 3 nodes of interest (dash-circled) and we would like to extract a relevant subgraph explaining the relationships between these 3 nodes.

Another typical application of the proposed methods is the traffic analysis of vehicles running between several locations. The road network can be modeled as a graph in which nodes represent locations. Each road between any pair of adjacent locations can be characterized by an edge weight. The higher the weight between two nodes the easier the immediate connection between them. A connection weight is typically inversely proportional to the time to travel from one node to the other or to the distance between them. According to the chosen modeling hypothesis, the corresponding weighted graph can either be directed or undirected. In the latter case, the weight also corresponds to the notion of *conductance* in an electrical network as detailed in section 8.7. While analyzing the traffic over the whole network, one would like to extract the most relevant routes to connect a predefined subset of $k$, non-necessarily adjacent, locations. The extracted subgraph should not only reflect the shortest path(s) between any pair of locations of interest but how the overall traffic can be distributed while connecting them.

Consider, for instance, the graph[3] depicted on Figure 8.2. The nodes represent particular locations on a roadmap. Edge weights correspond here to the inverse distance between nodes. Since distances are symmetric the graph is undirected. In this particular example, the weight of any (existing) edge is assumed to be equal to 1. For the simplicity of the reasoning, we first consider only two nodes of interest (1 and 9). They typically identify

[1] Several representations exist for metabolic networks (see *e.g.* [38]). The approach described here can be applied whether the network is represented as a directed or undirected graph, bipartite or not, and as long as it is connected. The unconnected case is briefly discussed in section 8.5.

[2] This figure has been kindly provided to us by J. Van Helden from the *Service de Conformation des Macromolécules Biologiques et de Bioinformatique* of the *Université Libre de Bruxelles*.

[3] This example is inspired from [95].

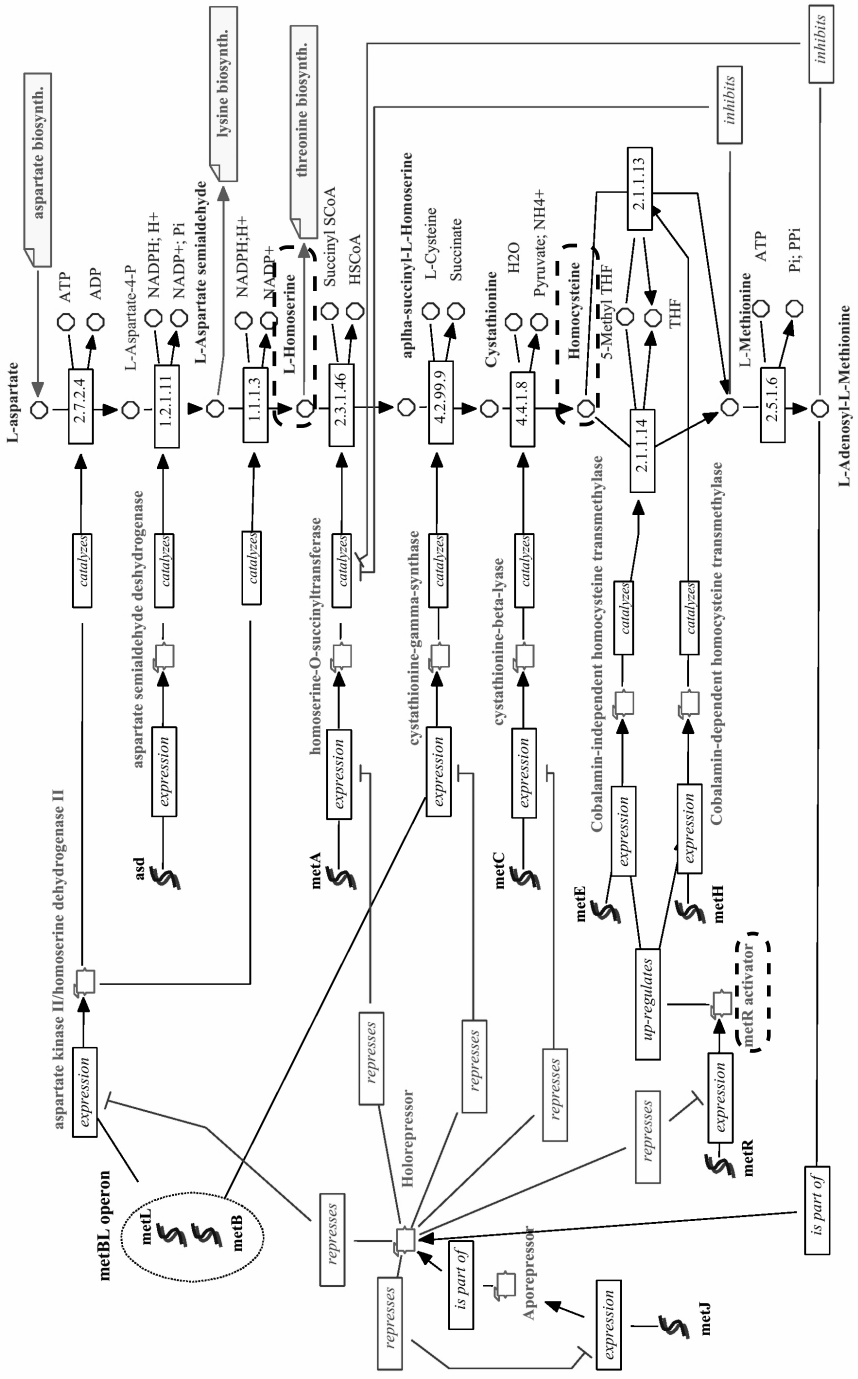



Figure 8.1: Methionine Biosynthesis of E. Coli.

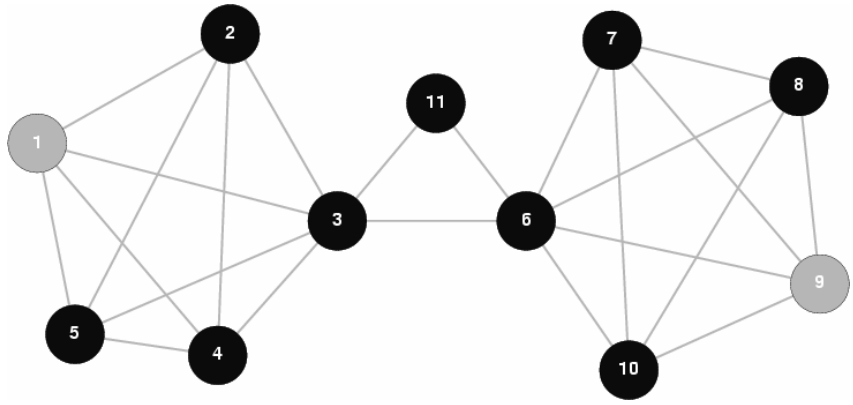



Figure 8.2: A roadmap describing the possible routes between a set of locations.

two distinct locations, each one belonging for example to a specific suburb (a highly connected cluster of locations). The best route between 1 and 9 corresponds here to the shortest distance path 1-3-6-9 or, equivalently, 9-6-3-1. Suppose we would like to know the second best route (in case of a potential traffic jam along the 3-6 edge, for instance). If one considers the second shortest path(s), there are many routes having the same total distance (equal to 4 in this case): 1-4-3-6-9, 1-3-6-8-9, 1-3-11-6-9, ... One can thus apply an algorithm for finding the $m$-shortest paths [47, 67] but this algorithm would consider all routes of length 4 as equally important. The unique second best route in this particular example is arguably 1-3-11-6-9 as it forms the unique best alternative to the direct edge 3-6 connecting two cut nodes. In other words, a random walker starting from node 1 and reaching eventually node 9 (or the converse) is more likely to go through node 11 than any other node as an alternative to the best route. Conversely, a random walker following the 1-2 edge instead of 1-3 is less likely to choose 2-3 as the next edge, simply because there are many other options to leave node 2. The proposed approach is precisely based on the expected number of times a given node or a given edge is used along any random walk connecting the nodes of interest. The resulting graph is depicted on Figure 8.3 where each edge width denotes its relative frequency of use along these walks. This relative frequency is interpreted as its *relevance* to explain the relationships between the nodes of interest. Discarding any edge the relevance of which falls below a threshold defines a relevant subgraph. A more stringent threshold would result in a smaller subgraph, *e.g.* the subgraph induced by the nodes $\{1, 3, 6, 9, 11\}$. The resulting subgraph obtained by such a thresholding needs not be connected however. This connectivity

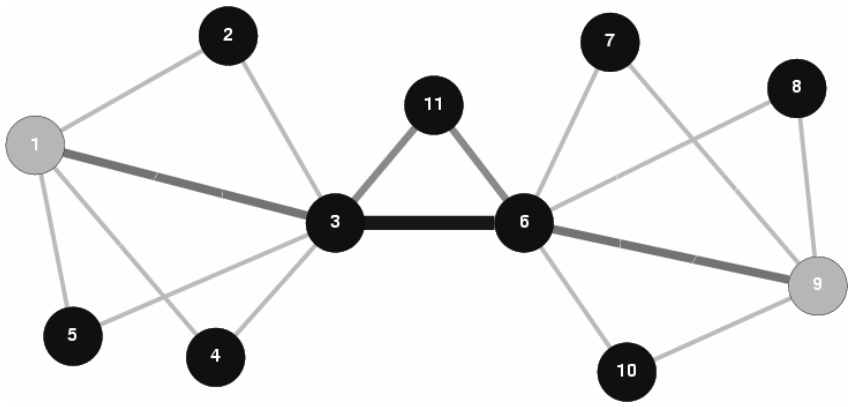



Figure 8.3: Relative edge relevance based on random walks between nodes 1 and 9.

issue is further discussed in section 8.5.

The random walk method proposed here also overcomes limitations introduced by a maximal flow approach. Consider for instance another roadmap[3] depicted on Figure 8.4. We are interested in relevant ways to connect nodes 1 and 8, each one belonging to a separate cluster[4]. All (existing) edge weights are again assumed to be equal to 1. These weights can also be interpreted as the flow capacity between any pair of adjacent nodes. There are alternatives routes to connect the two clusters but there is a maximal flow of 2 units from one cluster to the other. Nodes such as 12 or 15 can each one capture one unit of flow (for instance, from left to right) and thus would be considered as important to connect the two nodes (or clusters) of interest. Node 17 would not be considered as important since flowing through this node would limit the total flow between the two clusters to 1 unit. This is not necessarily appropriate since there are routes through 17 which can be considered as good alternatives to the "straight" routes. In particular, 1-3-11-17-16-10-8 also defines a relevant shortest path. Relative edge relevances based on the proposed random walk approach is depicted on Figure 8.5. The edges 3-11 and 16-10 are the most important since any left-to-right route between the two clusters must at least pick one of them, sometimes both. The edges 11-17 and 17-16 are non-negligible as they are part of important alternative routes.

The proposed approach relies on an interpretation of the graph as a Markov chain. The states of such a model correspond to the nodes in the

[4]These clusters form cliques in the present case but this is not a mandatory feature.

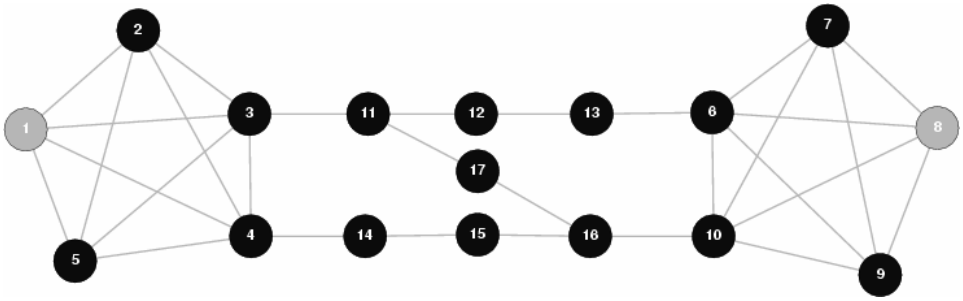



Figure 8.4: Another roadmap example.

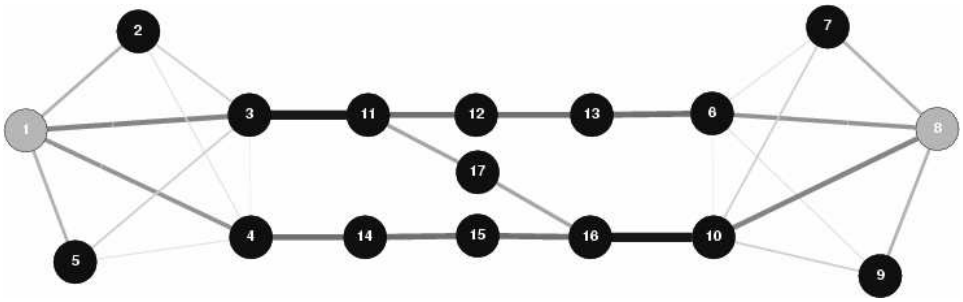



Figure 8.5: Relative edge relevance based on random walks between nodes 1 and 8.

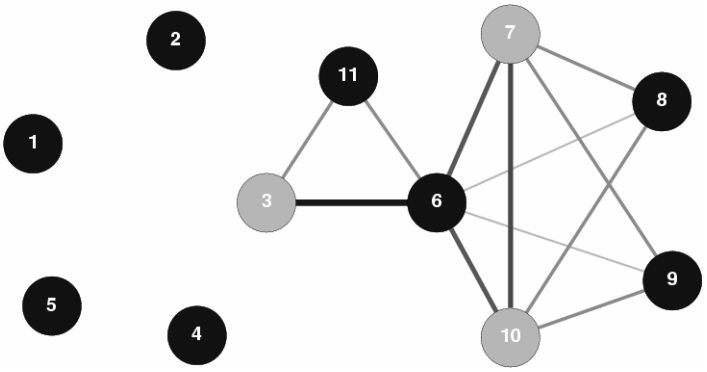



Figure 8.6: Edge relevances relatively to the nodes 3, 7 and 10.

original graph and the transition probabilities are defined proportionally to the edge weights. Markov chain theory [73, 98] allows one to compute the nodes and edges most frequently visited while performing random walks between any pair of distinct nodes of interest. The walks are random but the probability distribution over all possible walks is generally far from uniform. Hence the likelihood of any given walk actually matters in the relevance computation. These notions are formally presented in section 8.2 detailing the proposed *$\mathcal{K}$-walk approach*.

The edge or node relevance measures used here are *betweenness centrality indices* [17]. Shortest-path betweenness has been proposed in [51]. Random walk betweenness [95] was introduced more recently. Our approach can be considered as offering several extensions to Newman's work. In particular, we do not restrict our attention to undirected unweighted graphs. More importantly, edge and node relevances are here evaluated *relatively* to the choice of $k$ ($\geq 2$) nodes of interest rather than defined as a fixed characterization of a given graph. For instance, Figure 8.6 depicts the edge relevances of the graph of Figure 8.2 when nodes 3, 7 and 10 are selected as nodes of interest. The result is clearly distinct from the graph depicted on Figure 8.3.

Moreover, as pointed by Newman, shortest-path betweenness and random walk betweenness can be considered at the opposite end of a spectrum of possibilities in terms of the number of walk steps. Indeed a random walker has no idea of where he is heading to while a shortest-path walker intends to follow a straightest path from source to destination[5]. The *limited $\mathcal{K}$-walk approach*, presented in section 8.3 , can be considered as an intermediate along this spectrum. Here the walker travels also at random but only those walks of a prescribed length $L$ are considered. A slightly distinct variant

[5]Whenever unit weights are assigned to the edges, shortest distance paths are also minimal in number of steps.

considers all random walks up to a maximal length $L_{max}$. When $L_{max}$ tends to infinity this method becomes equivalent to the original $\mathcal{K}$-walk approach. Both algorithms are however distinct from a computational viewpoint as discussed in section 8.3. Section 8.4 describes another extension where the nodes of interest are no longer considered individually but as groups of *a priori* related nodes.

We discussed so far how to define a relevance measure on the edges and nodes of a graph to best explain the relationships between $k$ nodes of interest. Keeping only those edges whose relevance is above a prescribed threshold is a direct way to extract a non-necessarily connected subgraph. Section 8.5 elaborates on the subgraph extraction itself. A relevant subgraph should be, in many cases, as small as possible while capturing most of the information to explain the relationships between the nodes of interest. In other words, one would like to discriminate between important and less important edges (or nodes). The $\mathcal{K}$-walk or limited $\mathcal{K}$-walk approaches are not necessarily highly discriminant since their relevance measures are based on a large, possibly infinite, set of walks in the graph. Many walks in this set can largely overlap while partially sharing the captured information. Whether or not the extracted subgraph is small relative to the initial graph also depends on the graph topology, the initial edge weights and the chosen nodes of interest. In any case, one can enforce more discrimination by *inflating* the differences between the edges as detailed in section 8.6.

An inspiring approach to the problem of extracting a relevant subgraph is described in [48]. To the best of our knowledge, this is the only previous work where the problem of extracting a relevant subgraph from a given set of nodes of interest has been stated. The problem was however restricted to 2 nodes of interest in an undirected graph. In this approach, the 2 nodes of interest are respectively considered to be the source and the sink of an electrical current. The algorithm searches the paths followed by the current flow and maximizes the sum of current flow in the extracted subgraph. In addition, each node includes some current loss in order to penalize long paths and very highly connected nodes (hubs). A comparison with this approach is further discussed in section 8.7 where we present an electrical equivalent of the $\mathcal{K}$-walk approach for undirected graphs.

Let us finally mention that random walks and First Passage Times (FPT) have been used recently in a variety of applications in data mining and machine learning. For instance, Harel & Koren [59] investigates the possibility of clustering data according to some random-walk related quantities, such as the probability of visiting a node before returning to the starting node.

They show that their algorithm is able to cluster arbitrary nonconvex shapes. White & Smyth [134] investigates the use of the Mean First Passage Times (MFPT, see definition 18) as a similarity measure between nodes. Their purpose is to generalize the random-walk approach of Page et al. [18, 101] by capturing a concept of *relative centrality* of a given node with respect to some other node of interest. A recent study, comparing several measures (including MFPT) for analyzing the proximity of nodes in a graph in the framework of co-authorship networks, is presented in [85]. Commute Times (CT) during random walks are closely related to FPT: they count the number of steps a random walker, starting from a given node, will take before entering another given node for the first time, and go back to the starting node. The CT kernel is introduced in [112, 50] and is based on mean commute times during random walks. From the geometrical viewpoint, the CT kernel computes the dot product in a feature space where the nodes are exactly separated by their commute-time distance. Qiu & Hancock [105, 106], Ham, Lee, Mika & Scholkopf [58], Yen et al. [138] as well as Brand [15] apply the same CT embedding to image segmentation and multi-body motion tracking [105, 106], to dimensionality reduction of manifolds [58], to clustering [138] as well as to collaborative filtering [15], with good results. Besides, Zhou [141, 140] uses the MFPT between two nodes as a dissimilarity index in order to cluster them. He studies various greedy clustering techniques based on this dissimilarity index. An application of the CT distance to semi-supervised learning is proposed in [39].

## 8.2 $\mathcal{K}$-walks in a graph

Let $G = (\mathcal{V}, \mathcal{E})$ denote a graph formed by a set $\mathcal{V}$ of vertices (or nodes) and a set $\mathcal{E}$ of edges, with $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. Let $n_s = |\mathcal{V}|$ and $n_t = |\mathcal{E}|$ respectively denote the graph order and the number of edges in the graph. We consider in particular connected graphs represented by their weighted $n_s \times n_s$ adjacency matrix $M$. The $M_{qq'}$ entry of $M$ denotes the weight of the edge connecting node $q$ to node $q'$. Weights are assumed to be zero if and only if their corresponding edge does not belong to the graph. Otherwise, the weight between any pair of connected nodes is assumed strictly positive. Moreover edge weights should be defined such that the larger the weight $M_{qq'}$ the easier the connection (or communication or information flow) from $q$ to $q'$. We consider both directed and undirected graphs. An undirected graph is simply represented by a *symmetric* matrix $M$. In the directed case, the graph is assumed weakly connected and there must exist a directed path from any node to at least one node of interest. The diagonal degree matrix is defined as $D = \mathrm{Diag}(d_1, \ldots, d_{n_s})$ with $d_q = \sum_{q' \in \mathcal{V}} M_{qq'}$. Whenever $M$

reduces to a binary adjacency matrix (weights are either 0 or 1), $d_q$ simply denotes the degree[6] of node $q$. In general, $d_q$ is interpreted as the weighted degree of node $q$. A related quantity is the *graph volume* $D_G = \sum_{q \in \mathcal{V}} d_q$.

Given a weighted adjacency matrix $M$ associated to a graph $G = (\mathcal{V}, \mathcal{E})$ and a subset $\mathcal{K} \subseteq \mathcal{V}$ ($|\mathcal{K}| \geq 2$) of nodes of interest, we define relevance indices on the edges or nodes of $G$. They measure how much each node or edge contributes to the relationships between the nodes of $\mathcal{K}$. The number $k = |\mathcal{K}|$ of nodes of interest is typically much smaller than $n_s$. Formally, we define a *node relevance* function $nr_{\mathcal{K}} : \mathcal{V} \to \mathbb{R}^+$ which maps any node to its relevance. We define an *edge relevance* function $er_{\mathcal{K}} : \mathcal{E} \to \mathbb{R}^+$, similarly. As motivated in section 8.1, the relevances should rely on all possible ways to connect the $k$ nodes of interest (each way having a certain likelihood) and not only shortest-distance or maximal-flow paths. Technically, we propose to define the relevance of a node or an edge to be proportional to the *expected number of times* it is used when randomly walking through the graph, starting from one node of interest and eventually reaching a distinct node of interest.

Section 8.2.1 describes how the input graph is transformed into a Markov chain and introduces the relevance indices for nodes and edges. Section 8.2.2 details how these relevances are computed in practice.

### 8.2.1 A dynamic view of the graph

Random walks in a graph can be modeled by a Markov chain $T = \langle \mathcal{V}, A, \boldsymbol{\iota} \rangle$ (see definition 1) describing the sequence of nodes visited during the walk. The state set of the MC is the set of nodes $\mathcal{V}$, *i.e.* each state of the Markov chain corresponds to a distinct node of the graph. Hence, the terms *nodes* and *states* are used interchangeably in this chapter. The random variable $X_t$ represents the state of the MC reached at time $t$. The probability of transiting to state $q'$ at time $t+1$, given that the current state is $q$ at time $t$, is given by:

$$P[X_{t+1} = q' \mid X_t = q] = A_{qq'} = \frac{M_{qq'}}{d_q}. \tag{8.1}$$

Thus, from any state $q$, the (homogeneous) probability to jump to state $q'$ is proportional to the weight $M_{qq'}$ of the edge from $q$ to $q'$. The transition matrix $A = [A_{qq'}]$ of the Markov chain is related to the degree and adjacency matrices as $A = D^{-1}M$. Note that even when $M$ is symmetric, $A$ is generally asymmetric since the degrees of adjacent nodes need not be equal. $A$ is stochastic since, by construction, $0 \leq A_{qq'} \leq 1$ and $\sum_{j=1}^{n_s} A_{qq'} = 1$. The initial distribution $\boldsymbol{\iota}$ is considered as a user parameter, not derived from

[6] $d_q$ denotes the *outgoing* degree of node $q$ whenever the graph is directed.

the input graph. It is typically defined such that $\iota_x > 0 \Leftrightarrow x \in \mathcal{K}$. This probability distribution encodes some *prior* preference between the nodes of interest. By default, all nodes of interest are assumed equally important and $\iota_x = \frac{1}{k}, \forall x \in \mathcal{K}$.

The proposed approach is concerned with specific random walks beginning in a node of interest and ending when another node of interest is reached. These walks, called $\mathcal{K}$-walks, are formally defined hereafter.

**Definition 42 ($\mathcal{K}$-walk)** *Given a MC $T = \langle \mathcal{V}, A, \boldsymbol{\iota} \rangle$ and a set of nodes of interest $\mathcal{K}$ with $|\mathcal{K}| \geq 2$, a $\mathcal{K}$-walk is a word[7] $\nu = xwr$ defined on $\mathcal{V}$ such that $x, r \in \mathcal{K}$, $x \neq r$ and $w \in (\mathcal{V}_T)^*$ where $\mathcal{V}_T = (\mathcal{V} \setminus \mathcal{K}) \cup \{x\}$.*

In the sequel, the set of all $\mathcal{K}$-walks is denoted by $\mathcal{W}$ and the set of $\mathcal{K}$-walks starting in a specific state $x \in \mathcal{K}$ is denoted by ${}^x\mathcal{W}$. The relevance of a node $q \in \mathcal{V}$ is defined as the expectation of passage times $\mathrm{pt}(q)$ (see definition 15) in $q$ during $\mathcal{K}$-walks.

**Definition 43 (Node relevance)** *Given a MC $T = \langle \mathcal{V}, A, \boldsymbol{\iota} \rangle$ and a set of $\mathcal{K}$-walks $\mathcal{W}$, the* node relevance function *$nr_{\mathcal{K}} : \mathcal{V} \to \mathbb{R}^+$ is defined as*

$$nr_{\mathcal{K}}(q) = \mathbb{E}\left[\mathrm{pt}(q) \mid \mathcal{W}\right] \tag{8.2}$$

*for all $q \in \mathcal{V}$.*

The *edge passage times* is a function $\mathrm{ept} : \mathcal{E} \to \mathbb{R}$ computing the number of times an edge $q \times q'$ is triggered during random walks. The edge relevance function is defined according to the expected edge passage times.

**Definition 44 (Edge relevance)** *Given a MC $T = \langle \mathcal{V}, A, \boldsymbol{\iota} \rangle$ and a set of $\mathcal{K}$-walks $\mathcal{W}$, the* edge relevance function *$er_{\mathcal{K}} : \mathcal{E} \to \mathbb{R}^+$ is defined as*

$$er_{\mathcal{K}}(q, q') = \begin{cases} \mathbb{E}\left[\mathrm{ept}(q, q') \mid \mathcal{W}\right] & \text{if } G \text{ is directed} \\ \sum_{x \in \mathcal{K}} \iota_x \left| \mathbb{E}\left[\mathrm{ept}(q, q') \mid {}^x\mathcal{W}\right] - \mathbb{E}\left[\mathrm{ept}(q', q) \mid {}^x\mathcal{W}\right] \right| & \text{otherwise} \end{cases} \tag{8.3}$$

*for all $q, q' \in \mathcal{V}$.*

In the undirected case, the absolute difference guarantees that edge relevances are symmetric. It is also motivated by an electrical interpretation detailed in section 8.7. The practical computations of node and edge relevances are detailed in the next subsection.

[7] A word defined on the alphabet $\mathcal{V}$ corresponds to a sequence of states.

### 8.2.2 Relevance computation

To compute the node and edge relevances, we will resort to absorbing MC techniques detailed in section 1.3. We essentially need to compute the expected passage times in each node $q \in \mathcal{V}$ during $\mathcal{K}$-walks (see definition 42). This computation can be split according to the initial state $x$ of the walks. The desired expectations are then obtained as a weighted sum over all possible starting nodes $x \in \mathcal{K}$ (see equation (8.8)).

Given the MC $T$ obtained from the input graph G (see equation (8.1)), $\mathcal{K}$-walks can be studied by transforming all states in $\mathcal{K} \setminus \{x\}$ to be absorbing. Doing so prevents the walker to continue his walk when reaching a node of interest as he is getting trapped in such states. Let ${}^{x}A$ denote a modified transition matrix for which all nodes of interest but $x$ have been transformed to be absorbing. It is defined from $A$ as follows.

$$ {}^{x}A_{qq'} = \begin{cases} 1 & \text{if } q \in \mathcal{K} \setminus \{x\} \text{ and } q = q', \\ 0 & \text{if } q \in \mathcal{K} \setminus \{x\} \text{ and } q \neq q', \\ A_{qq'} & \text{otherwise.} \end{cases} \tag{8.4} $$

As the original graph is assumed to be connected[8], there is no absorbing state in the Markov chain defined by the original transition matrix $A$. Hence, only the states of interest but $x$ are absorbing according to ${}^{x}A$. The other states, including $x$ itself, form the set $\mathcal{V}_T$ of *transient* states from which there is a strictly positive probability to leave. The canonical block-partitioned form of ${}^{x}A$ is the following:

$$ {}^{x}A = \begin{pmatrix} {}^{x}F & {}^{x}E \\ 0 & I \end{pmatrix} \tag{8.5} $$

Here ${}^{x}F$ denotes the $(n_s - k + 1) \times (n_s - k + 1)$ transition submatrix between transient states, $I$ is the identity matrix of order $k - 1$ and ${}^{x}E$ is a $(n_s - k + 1) \times (k - 1)$ matrix. In particular, ${}^{x}E_{qr}$ denotes the probability of a walk being in a transient state $q$ to be absorbed in state $r \in \mathcal{K} \setminus \{x\}$ in one step. The probability to be absorbed in one step, no matter in which absorbing state, is computed by the $(n_s - k + 1)$-dimensional vector ${}^{x}E\mathbf{1}$; the $q$-th entry containing the absorption probability from the transient state $q$. Moreover, since $A$ is stochastic this vector can be obtained from the matrix ${}^{x}F$ as ${}^{x}E\mathbf{1} = (I - {}^{x}F)\mathbf{1}$.

[8] In the directed case, the graph is assumed weakly connected. Moreover, there must exist a path from any node to at least one node of interest. If, for a particular node of interest $x$, there exist only directed paths to itself but not to any other node of interest, we do not consider the transformed MC ${}^{x}A$ and simply iterate the construction over the other nodes of interest.

Theorem 3 (in section 1.3) can now be applied to compute the expected passage times in transient states. Importantly, it states that averaging the passage times over the possibly infinite set of random walks ending in an absorbing state can be performed, in a finite time, by computing the following matrix:

$$ {}^{x}N = (I - {}^{x}F)^{-1} \tag{8.6} $$

called the *fundamental matrix* of the absorbing MC. In particular, ${}^{x}N_{xq}$ is the expected number of times a walk starting in the node of interest $x$ goes through state $q$ before getting absorbed in any node of $\mathcal{K} \setminus \{x\}$, that is, before reaching a distinct node of interest:

$$ \mathbb{E}\left[\mathrm{pt}(q) \mid {}^{x}\mathcal{W}\right] = {}^{x}N_{xq} \tag{8.7} $$

Finally, the mean time to absorption $\mathrm{mta}(x)$ starting from $x$ (see definition 16) computes the expected length of $\mathcal{K}$-walks starting in $x$ and ending in any other node of interest. It is readily obtained from the fundamental matrix as $\mathrm{mta}(x) = [{}^{x}N\mathbf{1}]_x$.

The previous computations are restricted to $\mathcal{K}$-walks starting in a particular node of interest $x$. Considering $\mathcal{K}$-walks starting in any node of interest reduces to computing the same quantities separately for each possible starting node $x$ and to averaging these quantities according to the initial distribution $\boldsymbol{\iota}$. The node relevance function $nr_{\mathcal{K}} : \mathcal{V} \to \mathbb{R}^{+}$ (see definition 43) is computed as follows for all $q \in \mathcal{V}$ :

$$ nr_{\mathcal{K}}(q) = \mathbb{E}\left[\mathrm{pt}(q) \mid \mathcal{W}\right] = \begin{cases} \sum_{x \in \mathcal{K}} \iota_x \, {}^{x}N_{xq} & \text{if } q \in \mathcal{V} \setminus \mathcal{K}, \\ \iota_q \, {}^{q}N_{qq} & \text{otherwise.} \end{cases} \tag{8.8} $$

The sum in the above formula is replaced by a single term whenever $q$ is a node of interest since it is considered only once as a transient state, precisely when it is used as starting state.

The expected edge passage times of an edge $q \to q'$ during $\mathcal{K}$-walks starting in $x$ can be readily obtained from the expected passage times in state $q$ as follows:

$$ \mathbb{E}\left[\mathrm{ept}(q, q') \mid {}^{x}\mathcal{W}\right] = \begin{cases} {}^{x}N_{xq} \, {}^{x}A_{qq'} & \text{if } q \in \mathcal{V} \setminus \mathcal{K}, \\ {}^{q}N_{qq} \, {}^{q}A_{qq'} & \text{if } q = x \text{ and } q \in \mathcal{K}, \\ 0 & \text{if } q \neq x \text{ and } q \in \mathcal{K}. \end{cases} \tag{8.9} $$

When considering $\mathcal{K}$-walks starting from any state $x \in \mathcal{K}$, the expected edge passage times are obtained as follows for all $q, q' \in \mathcal{V}$:

$$ \mathbb{E}\left[\mathrm{ept}(q, q') \mid \mathcal{W}\right] = \sum_{x \in \mathcal{K}} \iota_x \, \mathbb{E}\left[\mathrm{ept}(q, q') \mid {}^{x}\mathcal{W}\right] \tag{8.10} $$

The edge relevance function $er_{\mathcal{K}} : \mathcal{E} \rightarrow \mathbb{R}^+$ can now be computed by applying definition 44 where the expected edge passage times are provided by equations (8.9) and (8.10).

To sum up, the application of the $\mathcal{K}$-walk approach to compute node and edge relevances essentially amounts to interpreting the graph as a Markov chain. The transition probability matrix $A$ of this chain can be computed in $\Theta(n_t)$ time with $n_t = |\mathcal{E}|$. Next, $k$ particular submatrices ${}^xF$ (one matrix for each node $x$ of interest) need to be considered. For each of them, the fundamental matrix ${}^xN$ is computed. This amounts to inverting the matrix $(I - {}^xF)$, which can be done in $\mathcal{O}(n_s^3)$. This is the core of the computational load of this approach. Node and edge relevances can be derived from the fundamental matrices in $\Theta(n_t)$ time. Overall, the time complexity of the $\mathcal{K}$-walk approach is $\mathcal{O}(kn_s^3)$.

## 8.3 Limited $\mathcal{K}$-walks in a graph

In the $\mathcal{K}$-walk approach, presented in section 8.2, any walk connecting the nodes of interest is considered, no matter its length. Alternatively, the relationships between the nodes of interest can be explained by walks of a fixed length $L$ or up to a maximal length $L_{max}$. This is the purpose of the limited $\mathcal{K}$-walk approach. Section 8.3.1 defines limited $\mathcal{K}$-walks and related relevances. Section 8.3.2 highlights the theoretical connection between the limited and the standard $\mathcal{K}$-walk approaches. Finally, section 8.3.3 presents an efficient technique to compute edge and node relevances.

### 8.3.1 Walk and relevance definitions

Formally, a limited $\mathcal{K}$-walk is defined as follows.

**Definition 45** *Given a MC $T = \langle \mathcal{V}, A, \boldsymbol{\iota} \rangle$, a set of nodes of interest $\mathcal{K}$ with $|\mathcal{K}| \geq 2$ and a strictly positive length $L$, a* limited $\mathcal{K}$-walk $\nu = xwr$ *is a $\mathcal{K}$-walk such that $|w| = L - 1$.*

Considering $\mathcal{K}$-walks up to a maximal length $L_{max}$ amounts to replacing the constraint $|w| = L - 1$ by $|w| < L_{max}$. The sets of $\mathcal{K}$-walks subject to these constraints are respectively denoted by $\mathcal{W}_L$ and $\mathcal{W}_{L_{max}}$.

Node and edge relevances are now defined according to the expected number of times they are visited during limited $\mathcal{K}$-walks. To compute these expectations, any node of interest $x$ is again considered in turn as the unique starting node in a transformed MC characterized by the ${}^xA$ transition matrix. The expected passage times of an edge $q \rightarrow q'$ can be computed as

follows:

$$\mathbb{E}\left[\text{ept}(q,q') \mid {}^{x}\mathcal{W}_L\right] = \sum_{l=0}^{L-1} \frac{P[X_l = q, X_{l+1} = q', {}^{x}\mathcal{W}_L]}{P[{}^{x}\mathcal{W}_L]} \tag{8.11}$$

In equation (8.11), $X_l$ is a random variable denoting the state visited at step $l$ of the walk. $P[{}^{x}\mathcal{W}_L]$ denotes the probability of performing a $\mathcal{K}$-walk of length $L$ starting in state $x$. Similarly, $P[X_l = q, X_{l+1} = q', {}^{x}\mathcal{W}_L]$ denotes the joint probability of visiting the edge $q \to q'$, between step $l$ and step $l+1$, and performing a $\mathcal{K}$-walk of length $L$ starting in state $x$. These probabilities can be computed from the ${}^{x}F$ and ${}^{x}E$ matrices associated to ${}^{x}A$ (see the block structure of ${}^{x}A$ described in equation (8.5)). In particular, the probability of a $\mathcal{K}$-walk of length $L$ starting in $x$ is given by

$$P[{}^{x}\mathcal{W}_L] = \sum_{r \in \mathcal{K}\setminus\{x\}} \left[({}^{x}F)^{L-1}\ ({}^{x}E)\right]_{xr}, \tag{8.12}$$

since such a walk transits $L-1$ times through transient states before getting absorbed in any state $r$ in $\mathcal{K} \setminus \{x\}$. The probability of visiting edge $q \to q'$ in such a walk, if $q'$ is a transient state and $l \leq L-2$, is given by

$$P[X_l = q, X_{l+1} = q', {}^{x}\mathcal{W}_L] = \sum_{r \in \mathcal{K}\setminus\{x\}} \left[({}^{x}F)^{l}\right]_{xq} [{}^{x}F]_{qq'} \left[({}^{x}F)^{L-l-2}\ ({}^{x}E)\right]_{q'r}. \tag{8.13}$$

Indeed, such a walk transits $l$ times through transient states while reaching $q$ in the $l^{\text{th}}$ step, transits from state $q$ to state $q'$ with probability $[{}^{x}F]_{qq'}$, transits again $L-l-2$ times through transient states before getting absorbed in any state $r$ in $\mathcal{K}\setminus\{x\}$. Whenever the destination state $q'$ of the edge $q \to q'$ is absorbing, the probability of visiting this edge is given by

$$P[X_{L-1} = q, X_L = q', {}^{x}\mathcal{W}_L] = \left[({}^{x}F)^{L-1}\right]_{xq} [({}^{x}E)]_{qq'}, \quad \forall q' \in \mathcal{K} \setminus \{x\} \tag{8.14}$$

since, for the walk length to be equal to $L$ before getting absorbed, such an edge can only be visited at the last step of the walk. The expected edge passage times during $\mathcal{K}$-walks of length $L$, starting in any state $x \in \mathcal{K}$ are computed as follows for all $q, q' \in \mathcal{V}$:

$$\mathbb{E}\left[\text{ept}(q,q') \mid \mathcal{W}_L\right] = \sum_{x \in \mathcal{K}} \iota_x\, \mathbb{E}\left[\text{ept}(q,q') \mid {}^{x}\mathcal{W}_L\right] \tag{8.15}$$

Considering $\mathcal{K}$-walks up to a maximal length $L_{\max}$ simply consists of a weighted sum of expectations over all these lengths:

$$\mathbb{E}\left[\text{ept}(q,q') \mid {}^{x}\mathcal{W}_{L_{max}}\right] \;=\; \sum_{L=1}^{L_{max}} \mathbb{E}\left[\text{ept}(q,q') \mid {}^{x}\mathcal{W}_L\right] P[{}^{x}\mathcal{W}_L] \tag{8.16}$$

Computing this expectation no matter the initial state $x \in \mathcal{K}$ is performed similarly as in equation (8.15).

Edge relevances $er_{\mathcal{K}}(q, q')$ are computed according to definition (44) where the expected edge passage times are replaced as follows. If one is interested only in the relationships between the nodes of interest for a fixed walk length $L$, the expectations $\mathbb{E}\left[\text{ept}(q, q') \mid \mathcal{W}\right]$ and $\mathbb{E}\left[\text{ept}(q, q') \mid {}^{x}\mathcal{W}\right]$ are respectively replaced by $\mathbb{E}\left[\text{ept}(q, q') \mid \mathcal{W}_L\right]$ and $\mathbb{E}\left[\text{ept}(q, q') \mid {}^{x}\mathcal{W}_L\right]$. For $\mathcal{K}$-walks up to length $L_{max}$, the former expectations are respectively replaced by $\mathbb{E}\left[\text{ept}(q, q') \mid \mathcal{W}_{L_{max}}\right]$ and $\mathbb{E}\left[\text{ept}(q, q') \mid {}^{x}\mathcal{W}_{L_{max}}\right]$.

Node relevances $nr_{\mathcal{K}}(q)$ are assigned according to the expected passage times $\mathbb{E}\left[\text{pt}(q) \mid {}^{x}\mathcal{W}_L\right]$ computed as the sum of the expected passage times on all outgoing edges from each node $q \in \mathcal{V}$:

$$nr_{\mathcal{K}}(q) = \sum_{x \in \mathcal{K}} \iota_x \mathbb{E}\left[\text{pt}(q) \mid {}^{x}\mathcal{W}_L\right], \tag{8.17}$$

$$\text{with } \mathbb{E}\left[\text{pt}(q) \mid {}^{x}\mathcal{W}_L\right] = \sum_{q' \in \mathcal{V}} \mathbb{E}\left[\text{ept}(q, q') \mid {}^{x}\mathcal{W}_L\right]. \tag{8.18}$$

When considering $\mathcal{K}$-walks up to length $L_{max}$, a weighted sum of the expectation $\mathbb{E}\left[\text{pt}(q) \mid {}^{x}\mathcal{W}_L\right]$ over all these lengths is performed, similarly as in equation (8.16).

### 8.3.2 Links between standard and limited $\mathcal{K}$-walks

We establish here a theoretical connection between limited $\mathcal{K}$-walks up to a maximal length $L_{max}$ and standard $\mathcal{K}$-walks introduced in section 8.2. More precisely, we show that both approaches become equivalent when letting $L_{max}$ tend to infinity. First, let us observe that when $L_{max}$ tends to infinity, the constraint $|w| < L_{max}$ (see definition 45 and below) becomes inactive and therefore $\lim_{L_{max} \to \infty} \mathcal{W}_{L_{max}} = \mathcal{W}$. Proposition 8 shows formally that the expected edge passage times for both approaches are equal when $L_{max}$ tends to infinity. Consequently, both approaches define asymptotically the same relevance measure.

**Proposition 8 (Convergence of limited $\mathcal{K}$-walks)**

$$\begin{aligned} \sum_{L=1}^{\infty} \mathbb{E}\left[\text{ept}(q, q') \mid {}^{x}\mathcal{W}_L\right] \; P[{}^{x}\mathcal{W}_L] &= \mathbb{E}\left[\text{ept}(q, q') \mid {}^{x}\mathcal{W}\right] \\ &= {}^{x}N_{xq} \, {}^{x}A_{qq'} \end{aligned} \tag{8.19}$$

**Proof.**

*Case 1:* $q' \notin \mathcal{K} \setminus \{x\}$

$$\sum_{L=1}^{\infty} \mathbb{E}\left[\text{ept}(q,q') \mid {}^{x}\mathcal{W}_L\right] \; P[{}^{x}\mathcal{W}_L] =$$

$$\begin{aligned}
& \sum_{L=1}^{\infty} P[{}^{x}\mathcal{W}_L] \sum_{l=0}^{L-2} \frac{\sum_{r\in\mathcal{K}\setminus\{x\}} \left[({}^{x}F)^{l}\right]_{xq} \left[{}^{x}F\right]_{qq'} \left[({}^{x}F)^{L-l-2}\; ({}^{x}E)\right]_{q'r}}{P[{}^{x}\mathcal{W}_L]} \\
= \; & \left[{}^{x}F\right]_{qq'} \sum_{L=1}^{\infty} \sum_{l=0}^{L-2} \left[({}^{x}F)^{l}\right]_{xq} \left[({}^{x}F)^{L-l-2}(I - {}^{x}F)\mathbf{1}\right]_{q'} \\
= \; & \left[{}^{x}F\right]_{qq'} \sum_{l=0}^{\infty} \left[({}^{x}F)^{l}\right]_{xq} \sum_{L=l+2}^{\infty} \left[({}^{x}F)^{L-l-2}(I - {}^{x}F)\mathbf{1}\right]_{q'} \\
= \; & \left[{}^{x}F\right]_{qq'} \left[(I - {}^{x}F)^{-1}\right]_{xq} \left[(I - {}^{x}F)^{-1}(I - {}^{x}F)\mathbf{1}\right]_{q'} \\
= \; & \left[{}^{x}F\right]_{qq'} \; {}^{x}N_{xq} = {}^{x}N_{xq} \; {}^{x}A_{qq'}
\end{aligned}$$

*where* $\mathbf{1}$ *denotes a* $(n_s - k + 1)$*-dimensional vector of ones.*

*Case 2:* $q' \in \mathcal{K} \setminus \{x\}$

$$\begin{aligned}
\sum_{L=1}^{\infty} \mathbb{E}\left[\text{ept}(q,q') \mid {}^{x}\mathcal{W}_L\right] \; P[{}^{x}\mathcal{W}_L] \; &= \; \sum_{L=1}^{\infty} P[{}^{x}\mathcal{W}_L] \frac{\left[({}^{x}F)^{L-1}\right]_{xq} \left[({}^{x}E)\right]_{qq'}}{P[{}^{x}\mathcal{W}_L]} \\
&= \; \left[({}^{x}E)\right]_{qq'} \sum_{L=1}^{\infty} \left[({}^{x}F)^{L-1}\right]_{xq} \\
&= \; \left[({}^{x}E)\right]_{qq'} \; {}^{x}N_{xq} = {}^{x}N_{xq} \; {}^{x}A_{qq'}
\end{aligned}$$

■

This result has a practical consequence as it offers an alternative way to compute the non limited $\mathcal{K}$-walks. This directly suggests a way to approximate standard $\mathcal{K}$-walks by considering large but finite $L_{max}$ values. As detailed, in section 8.2.2 the standard $\mathcal{K}$-walk has a time complexity $\mathcal{O}(kn_s^3)$ whereas the limited $\mathcal{K}$-walk approach has a time complexity $\Theta(kn_tL_{max})$ (see section 8.3.3) where $k$ is the number of nodes of interest and $n_s, n_t$ are respectively the number of nodes and edges in the input graph. Therefore, when considering moderate $L_{max}$ values, the limited approach offers a significant speed-up with respect to the standard approach. Moreover, we can show theoretically that the limited $\mathcal{K}$-walk approach converges *almost geometrically fast* to the standard $\mathcal{K}$-walk approach. To do so, let us define the approximation error made when performing the limited $\mathcal{K}$-walk up to length

$L_{max}$. We distinguish the two cases considered in the proof of Proposition 8. The approximation error for the *Case 1* (*i.e.* q' is a transient state) is

$$\begin{aligned}\mathrm{Err}^{C1}_{L_{max}}(x,q,q') &= \sum_{L=L_{max}+1}^{\infty} \mathbb{E}\left[\mathrm{ept}(q,q') \mid {}^{x}\mathcal{W}_L\right]\ P[{}^{x}\mathcal{W}_L] \\ &= [{}^{x}F]_{qq'} \sum_{L=L_{max}+1}^{\infty} \textstyle\sum_{l=0}^{L-2} \left[({}^{x}F)^{l}\right]_{xq} \left[({}^{x}F)^{L-l-2}\boldsymbol{e})\right]_{q'}\end{aligned} \tag{8.20}$$

where $\boldsymbol{e} = (I - {}^{x}F)\mathbf{1}$. An upper bound on the approximation error can be obtained by individually bounding the entries $\left[({}^{x}F)^{l}\right]_{xq}$ and $\left[({}^{x}F)^{L-l-2}\boldsymbol{e})\right]_{q'}$ using a spectral decomposition of ${}^{x}F$. Assuming that the matrix ${}^{x}F$ is diagonalizable, the spectral decomposition of its $l$-th power is given by

$${}^{x}F^{l} = \lambda_1^{l}\boldsymbol{r}_1\boldsymbol{l}_1^{T} + \lambda_2^{l}\boldsymbol{r}_2\boldsymbol{l}_2^{T} + \ldots + \lambda_R^{l}\boldsymbol{r}_R\boldsymbol{l}_R^{T}, \tag{8.21}$$

where $\lambda_i$ is the $i$-th largest eigenvalue in modulus, $\boldsymbol{r}_i$ and $\boldsymbol{l}_i$ are respectively the right-hand and left-hand eigenvectors associated with $\lambda_i$ and $R \leq n_s - k + 1$ is the rank of ${}^{x}F$. Using this spectral decomposition, we can bound $\left[({}^{x}F)^{l}\right]_{xq}$:

$$\begin{aligned}\left[({}^{x}F)^{l}\right]_{xq} &= \lambda_1^{l}(\boldsymbol{r}_1\boldsymbol{l}_1^{T})_{xq} + \lambda_2^{l}(\boldsymbol{r}_2\boldsymbol{l}_2^{T})_{xq} + \ldots + \lambda_R^{l}(\boldsymbol{r}_R\boldsymbol{l}_R^{T})_{xq} \\ &\leq \lambda_1^{l}\, U_{xq}\end{aligned} \tag{8.22}$$

where $U_{xq} = |(\boldsymbol{r}_1\boldsymbol{l}_1^{T})_{xq}| + |(\boldsymbol{r}_2\boldsymbol{l}_2^{T})_{xq}| + \ldots + |(\boldsymbol{r}_R\boldsymbol{l}_R^{T})_{xq}|$. Similarly, one can bound the second considered entry:

$$\begin{aligned}\left[({}^{x}F)^{L-l-2}\boldsymbol{e})\right]_{q'} &= \textstyle\sum_{r=1}^{n_s-k+1} \left[({}^{x}F)^{L-l-2}\right]_{q'r} e_r \\ &\leq \textstyle\sum_{r=1}^{n_s-k+1} \lambda_1^{L-l-2} U_{q'r} e_r \\ &= \lambda_1^{L-l-2}\, \overline{U}_{q'}\end{aligned} \tag{8.23}$$

where $\overline{U}_{q'} = \sum_{r=1}^{n_s-k+1} U_{q'r}e_r$. The approximation error can now be bounded by

$$\begin{aligned}\mathrm{Err}^{C1}_{L_{max}}(x,q,q') &\leq [{}^{x}F]_{qq'}\, U_{xq}\, \overline{U}_{q'} \sum_{L=L_{max}+1}^{\infty} (L-1)\lambda_1^{L-2} \\ &= [{}^{x}F]_{qq'}\, U_{xq}\, \overline{U}_{q'} \left(\tfrac{\lambda_1}{(1-\lambda_1)^2} + \tfrac{1}{1-\lambda_1} L_{max}\right) \lambda_1^{L_{max}-1}\end{aligned} \tag{8.24}$$

A similar reasoning can be applied for the *Case 2* (*i.e.* q' is an absorbing state), leading to the following bound on the approximation error:

$$\begin{aligned}\mathrm{Err}^{C2}_{L_{max}}(x,q,q') &= [({}^{x}E)]_{qq'} \sum_{L=L_{max}+1}^{\infty} \left[({}^{x}F)^{L-1}\right]_{xq} \\ &\leq [({}^{x}E)]_{qq'}\, \tfrac{1}{1-\lambda_1} \lambda_1^{L_{max}}\end{aligned} \tag{8.25}$$

In section 8.8, we show that in practice small $L_{max}$ values, *e.g.* 50, already provide a good approximation.

### 8.3.3 Efficient relevance computation

Practical computation of the limited $\mathcal{K}$-walks can be performed efficiently via forward and backward recurrences similar to those used in the expectation steps of the PHIT (see section 7.3.1) and POMMPHIT (see section 5.5.2) algorithms.

Given a state $q \in \mathcal{V}$ and a time $l \in \mathbb{N}$, the forward variable $\alpha^{\mathcal{K}}(q,l)$ computes the probability of being in state $q$ after $l$ steps while starting in state $x$.

$$\alpha^{\mathcal{K}}(q,l) = P[X_l = q, \{X_k\}_{k=1}^{l-1} \in \mathcal{V}_T \mid X_0 = x] \tag{8.26}$$

The forward variables are computed using the following recurrence for $q \in \mathcal{V}$ and $l \in \mathbb{N}$:

$$\begin{array}{lrcl} \text{(Base case)} & \alpha^{\mathcal{K}}(q,0) & = & \begin{cases} 1 & \text{if } q = x \\ 0 & \text{otherwise} \end{cases} \\ \text{(Induction)} & \alpha^{\mathcal{K}}(q,l) & = & \sum_{q' \in \mathcal{V}_T} \alpha^{\mathcal{K}}(q', l-1)[{}^{x}A]_{q'q} \end{array} \tag{8.27}$$

Note that if $q$ is a transient state, $\alpha^{\mathcal{K}}(q,l)$ actually computes $\left[({}^{x}F)^{l}\right]_{xq}$ whereas if $q$ is an absorbing state it corresponds to $\left[({}^{x}F)^{l-1}\ {}^{x}E\right]_{xq}$.

Given a state $q \in \mathcal{V}$ and a time $l \in \mathbb{N}$, the backward variable $\beta^{\mathcal{K}}(q,t)$ computes the probability that state $q$ is reached by the process $t$ steps before getting absorbed:

$$\beta^{\mathcal{K}}(q,l) = P[X_0 = q, \{X_k\}_{k=1}^{l-1} \in \mathcal{V}_T \mid X_l \in \mathcal{K} \setminus \{x\}] \tag{8.28}$$

The following recurrence computes the backward variables for $q \in \mathcal{V}$ and $l \in \mathbb{N}$:

$$\begin{array}{lrcl} \text{(Base case)} & \beta^{\mathcal{K}}(q,0) & = & \begin{cases} 1 & \text{if } q \in \mathcal{K} \setminus \{x\} \\ 0 & \text{otherwise} \end{cases} \\ \text{(Induction)} & \beta^{\mathcal{K}}(q,l) & = & \begin{cases} 0 & \text{if } q \in \mathcal{K} \setminus \{x\} \\ \sum_{q' \in \mathcal{V}} \beta^{\mathcal{K}}(q', l-1)^{x}[A]_{qq'} & \text{otherwise} \end{cases} \end{array} \tag{8.29}$$

Note that $\beta^{\mathcal{K}}(q,l)$ actually computes the probability $\sum_{r \in \mathcal{K} \setminus \{x\}} \left[({}^{x}F)^{l-1}\ ({}^{x}E)\right]_{qr}$ $= P[{}^{q}\mathcal{W}_l]$.

The forward and backward recurrences are efficiently computed using an $n_t \times L_{max}$ lattice structure. The time complexity of the forward and backward recurrences for one transformed Markov chain is $\Theta(n_t L_{max})$, with $n_t = |\mathcal{E}|$. Equation (8.11) can be reformulated using these recurrences:

$$\mathbb{E}\left[\mathrm{ept}(q,q') \mid {}^{x}\mathcal{W}_L\right] = \frac{\sum_{l=0}^{L-1} \alpha^{\mathcal{K}}(q,l)\ [{}^{x}A]_{qq'}\ \beta^{\mathcal{K}}(q', L-l-1)}{\beta^{\mathcal{K}}(x,L)} \tag{8.30}$$

Equation (8.30) can be evaluated, from the two lattices, in $\Theta(n_t L)$ which, when summed over all possible lengths up to $L_{max}$ (equations (8.16) and (8.17)), can also be performed globally in $\Theta(n_t L_{max})$. Finally, as the above computations need to be repeated for $k$ Markov chains, the overall time complexity of the limited $\mathcal{K}$-walk approach is $\Theta(k n_t L_{max})$. An equivalent upper bound is $\mathcal{O}(k n_s^2 L_{max})$, but this upper bound is tight only if the graph is dense.

## 8.4 Groups of nodes of interest

The nodes of interest were considered so far independently of each other. In a more general setting, the elements of $\mathcal{K}$ can be partitioned into $p$ clusters: $\mathcal{K} = \{C_1, \ldots, C_p\}$ with $2 \leq p \leq k$. The node or edge relevances should now explain the relationships between the clusters rather than between the individual nodes. If the $\mathcal{K}$-walk approach is run while ignoring the cluster structure, node and edge relevances depend on the relationships between nodes of interest belonging to the same cluster, which is generally inappropriate. An alternative would be to construct a modified graph in which all nodes belonging to the same cluster are merged. This modified graph can however be an overly simplified view of the original graph.

A simple extension of the $\mathcal{K}$-walk approach considers the cluster structure explicitly. A transformed Markov chain is built relatively to each cluster rather than each node of interest. When the cluster $C_x$ is considered, all nodes belonging to $\mathcal{K} \setminus C_x$ are transformed to be absorbing. Hence the walks start in any node of $C_x$ and end in any node of $\mathcal{K} \setminus C_x$. Let ${}^{C_x}N$ denote the fundamental matrix of this transformed Markov chain. The mean node passage times are now defined as:

$$\forall q \in \mathcal{V}, nr_{\mathcal{K}}(q) = \begin{cases} \sum_{x \in \mathcal{K}} \iota_x\ {}^{C_x}N_{xq} & \text{if } q \in \mathcal{V} \setminus \mathcal{K}, \\ \iota_q\ {}^{C_q}N_{qq} & \text{otherwise.} \end{cases} \tag{8.31}$$

The expected edge passage times are defined analogously (see equation (8.9)). A similar reasoning can also apply to the limited $\mathcal{K}$-walk approach.

## 8.5 Subgraph extraction

The $\mathcal{K}$-walk and limited $\mathcal{K}$-walk approaches essentially amount to defining edge (or node) relevances proportionally to their frequencies of use along walks connecting the nodes of interest. A *relevant subgraph* can subsequently be extracted in several ways described hereafter.

The simplest approach requires a positive *edge relevance threshold* $\Theta_e$ and includes the edge $q \to q'$ in the extracted subgraph whenever $er_{\mathcal{V}}(q, q') > \Theta_e$. This procedure defines a subgraph induced by the selected edges, the larger $\Theta_e$ the smaller the induced subgraph. A positive *node relevance threshold* $\Theta_n$ can also be defined to include the node $q$ whenever $nr_{\mathcal{V}}(q) > \Theta_n$. The use of edge and node thresholds gives more flexibility. In particular, a specific node may be selected even though none of its incident edges are selected. Whether such a situation is desirable depends on the application context.

In the $\mathcal{K}$-walk approach, the extracted subgraph needs not be (weakly) connected even when thresholding is applied only on the edges. Whenever connectedness is required, one can search for the maximal $\Theta_e$ such that the induced subgraph is connected. Since the original graph is assumed connected, there must exist a critical threshold $\Theta_e^*$ such that the extracted subgraph is connected for any $\Theta_e \leq \Theta_e^*$. Automatically fixing $\Theta_e$ to this critical value defines a parameter-free subgraph extraction approach. The above approach is a greedy edge selection procedure which is efficient and locally minimal in terms of the number of edges of the induced subgraph. In the undirected case, one could also define edge costs inversely proportional to their respective relevances and look for the subgraph connecting the nodes of interest with a minimal total edge cost. This is a particular instance of the *Steiner tree problem* for undirected weighted graphs [63]. The optimal subgraph is indeed necessarily a tree in this case. This problem was shown NP-complete [53] but exact algorithms exist in restricted cases [133].

The limited $\mathcal{K}$-walk approach offers another alternative to construct a connected subgraph. Only the subset of walks of a given length $L$, or up to a maximal length $L_{max}$, are considered. In any case, a given edge relevance is strictly positive if and only if it is used in at least one walk of this subset. One can thus simply discard all edges with a null relevance. The resulting subgraph, if not empty, is connected by construction. It is also the smallest subgraph representing all walks of the prescribed length(s) between the nodes of interest. So far, the (limited) $\mathcal{K}$-walk approaches require the original graph to be connected. In the directed case, the graph is assumed weakly connected and there must exist a directed path from any node to at

least one node of interest. A possible extension would consider separately each connected component of the original directed or undirected graph and restrict the analysis in each component to the nodes of interest belonging to it.

## 8.6 Inflation

The $\mathcal{K}$-walk approach offers a global view of the ways nodes of interest are connected between each other. It is however not necessarily very discriminative between highly relevant edges (or nodes) versus less relevant ones. The relevance measures are indeed based on a large, possibly infinite, set of walks. Many of these walks overlap at least partially and the captured information is spread between them. This may be not optimal if the goal is to extract a small subgraph summarizing most of this information.

To some extent the limited $\mathcal{K}$-walk approach is more discriminative, especially for low $L_{max}$. In particular, when $k = 2$, if $L_{max}$ is fixed to the minimal number of steps between the 2 nodes of interest, only nodes or edges belonging to shortest paths, in terms of number of steps, have a strictly positive relevance. The larger $L_{max}$ the more walks are considered with a less discriminant result. Tuning $L_{max}$ is thus a way to control discrimination. In general, the discriminative character of the $\mathcal{K}$-walk approach (or the limited $\mathcal{K}$-walk approach for a large $L_{max}$) depends on the edge weights in the original graph, the chosen nodes of interest and the graph topology. If necessary, it is easy to inflate the differences between edge (or node) relevances by applying the $\mathcal{K}$-walk approach recursively. This notion of inflation is inspired from [128] where it used for defining graph clusters. The very simple way inflation is implemented in our approach is however different.

Figure 8.7 illustrates this idea. In the original graph on top, edge weights are assumed equal to 1. The graph obtained after one execution of the $\mathcal{K}$-walk approach is depicted in the middle. Here, the edge widths are their relative edge relevance. Edge relevances can be considered as new weights associated to edges and the same approach can be applied recursively. The result obtained after the second iteration is depicted at the bottom of Figure 8.7.

## 8.7 An electrical interpretation

An undirected graph is represented by a symmetric weighted adjacency matrix $M$. In this case, the graph can be seen as an electrical network, the

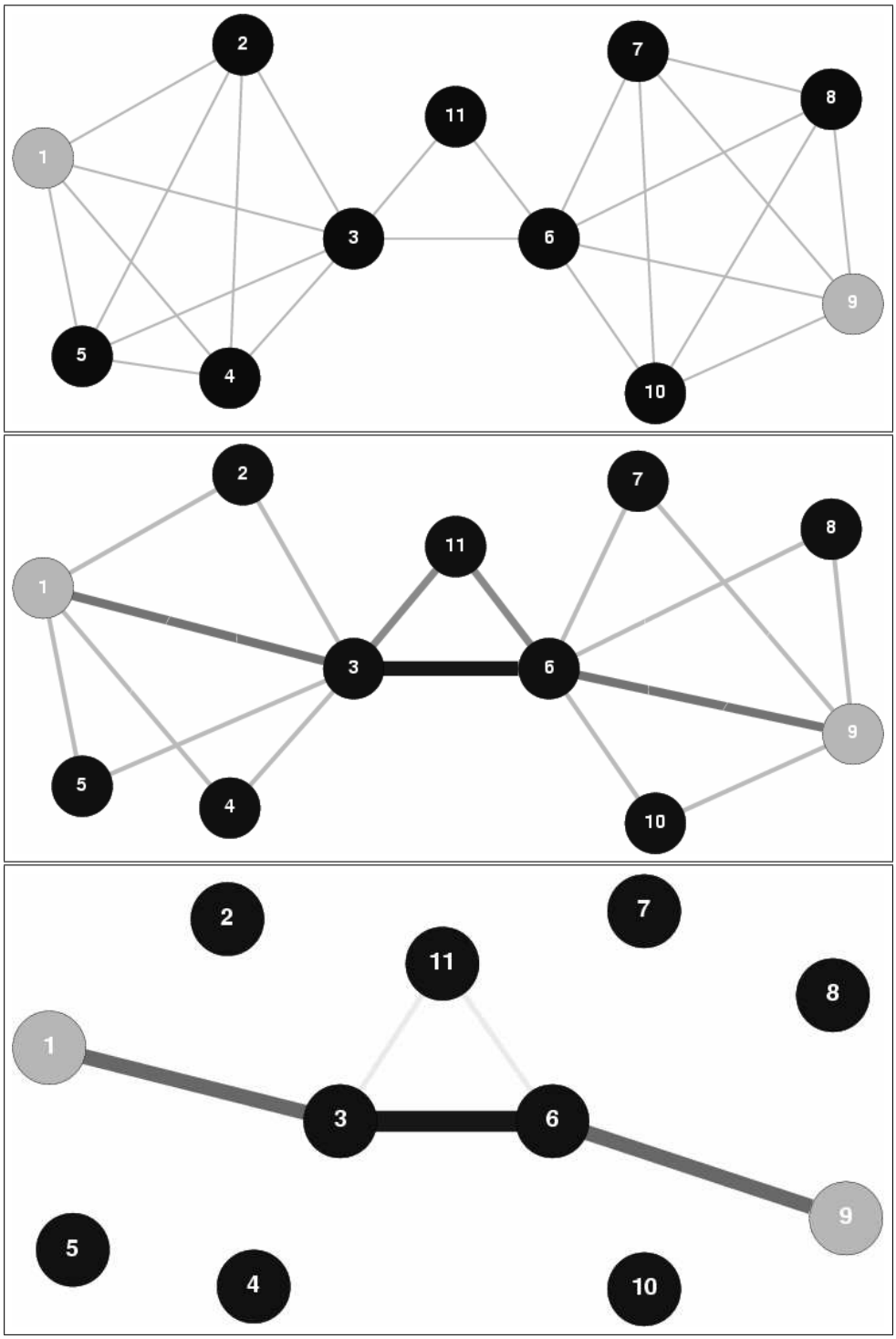



Figure 8.7: Inflating the relative edge relevances.

weight $M_{qq'}$ being the conductance between nodes $q$ and $q'$ [40]. According to equation (8.1), this symmetry implies a relationship between the transition probabilities (in the associated Markov chain) assigned to each direction for traversing an edge:

$$d_q A_{qq'} = d_{q'} A_{q'q}. \tag{8.32}$$

The degree $d_q$ is now interpreted as the sum of the conductances of the edges incident to node $q$.

When a node of interest $x$ is used as starting state of the walks, the network is represented by a transformed Markov chain with transition matrix ${}^{x}A$. Positively charged particles are assumed to enter the network at node $x$ and to leave the network at any node in $\mathcal{K}\setminus\{x\}$. The node $x$ is thus assumed to be the source of an electrical current and the other nodes of interest form current sinks. It could be even more realistic to consider negatively charged particles flowing in the opposite direction towards $x$ but the reasoning below is essentially equivalent. The currents observed between any nodes $q$ and $q'$ are directly related to the edge passage times in the $\mathcal{K}$-walk approach as described below.

By Ohm's Law, the current $\mathcal{I}_{qq'}$ from $q$ to $q'$ is determined by the voltages $V_q$ and $V_{q'}$ and the conductance between $q$ and $q'$:

$$\mathcal{I}_{qq'} = (V_q - V_{q'}) M_{qq'} \tag{8.33}$$

Kirchhoff's Current Law requires that the total current flowing through any node (but the current source and sinks) is 0:

$$\forall q, q' \in \mathcal{V}\setminus\mathcal{K},\ \sum_{q'} \mathcal{I}_{qq'} = 0. \tag{8.34}$$

The relationships between node voltages follows from the combination of equations (8.33) and (8.34):

$$\forall q, q' \in \mathcal{V}\setminus\mathcal{K},\ V_q = \sum_{q'} V_{q'}\ {}^{x}A_{qq'} \tag{8.35}$$

In the $\mathcal{K}$-walk approach, ${}^{x}N_{xq}$ is the expected number of times node $q$ is visited for a walk starting in $x$. This quantity can also be defined from the expected number of times any node $q'$ is visited:

$${}^{x}N_{xq} = \sum_{q'} {}^{x}N_{xq'}\ {}^{x}A_{q'q} \tag{8.36}$$

The relationship between these expected times and the voltages follows from the combination of equations (8.32) and (8.36):

$$\forall q, q' \in \mathcal{V}\setminus\mathcal{K},\ \frac{{}^{x}N_{xq}}{d_q} = \sum_{q'} \frac{{}^{x}N_{xq'}}{d_{q'}}\ {}^{x}A_{qq'} \tag{8.37}$$

Equations (8.35) and (8.37) are exactly equivalent when $V_q = \frac{{}^{x}N_{xq}}{d_q}$ showing that the voltages in this electrical network correspond to the node passage times normalized by the node degrees. Moreover, by equation (8.33), the current from $q$ to $q'$ is given by

$$\mathcal{I}_{qq'} = (\frac{{}^{x}N_{xq}}{d_q} - \frac{{}^{x}N_{xq'}}{d_{q'}})M_{qq'} = {}^{x}N_{xq}\ {}^{x}A_{qq'} - {}^{x}N_{xq'}\ {}^{x}A_{q'q} \tag{8.38}$$

The current $\mathcal{I}_{qq'}$ is the current along the edge $q \to q'$ actually flowing from $q$ to $q'$ whenever it is positive, or flowing in the opposite direction otherwise. Hence the net current along the edge $q \to q'$, independently of its direction, is the absolute value $|\mathcal{I}_{qq'}|$. This absolute value is exactly the net edge passage times considered in equation (8.3) in the case of an undirected graph. The edge relevances in the $\mathcal{K}$-walk approach correspond to the sum of the net currents obtained for $k$ electrical networks when each node $x$ of interest is used in turn as a current source of $\iota_x$ current unit.

The electrical interpretation of the $\mathcal{K}$-walk method illustrates some links and differences with the method proposed in [48]. The latter is restricted to 2 nodes of interest, respectively called the source and the destination, in an undirected graph. A one unit voltage is applied to the source and the destination is grounded. The voltages in all other nodes are computed by solving the system of linear equations (8.35). The electrical analogy is slightly different in the $\mathcal{K}$-walk approach since a unit current is injected in the source node rather than fixing its voltage. The respective currents and voltages are however identical in both electrical circuits up to a multiplicative constant equal to the effective resistance of the network [40]. In other words, the expected edge passage times are identical in relative terms, for one starting node. The $\mathcal{K}$-walk approach generalizes this methodology to any number of nodes of interest by considering $k$ electrical networks.

Both approaches also differ when considering the actual subgraph extraction. The algorithm proposed by Faloutsos et al. searches the paths followed by the current flow, from source to destination, and maximizes the sum of current flow in the extracted subgraph. This approach is based on the extraction of successive paths using dynamic programming. Such an approach is more difficult to extend to any number of nodes of interest since the paths to consider should be defined according to the $k$ nodes. In contrast, the $\mathcal{K}$-walk approach does not compute explicitly some paths but defines edge relevance according to their use in all possible walks. Walks, as opposed to paths, also include possible round trips along a given edge but the edge relevances are defined, for an undirected graph, as the net differences in both directions. Note that if the graph is directed or, more precisely, if $M$ is no longer symmetric, the electrical analogy presented here

does not hold anymore since physical laws require the conductance to be symmetric. The $\mathcal{K}$-walk approach can still be applied however.

A universal sink node, which absorbs a fraction of the current delivered to each node, is also introduced in [48]. The purpose of this current loss is to penalize very long paths between the two nodes of interest and paths going through highly connected nodes (hubs). The (limited) $\mathcal{K}$-walk approaches do not include such current losses. Discarding very long walks is the purpose of the limited $\mathcal{K}$-walks approach. We argue that this is a more direct and principled way of limiting the maximal walk length considered between the nodes of interest. Discarding hubs can be obtained by defining the weighted adjacency matrix in such a way that any edge connected to a hub has a low conductance.

Finally, solving the system of linear equations (8.35) can be done in $\mathcal{O}(n_s^3)$ but iterative methods can often perform faster for sparse graphs. As detailed in section 8.3, the limited $\mathcal{K}$-walk approach can be seen as an approximation to the $\mathcal{K}$-walk approach for a large $L_{max}$. The resulting time complexity reduces to $\Theta(n_t L_{max})$ for a single source node. The possible sparsity of the graph directly pays off here since this complexity is linear with $n_t$, the number of graph edges.

## 8.8 Experiments

An experimental study of the (limited) $\mathcal{K}$-walk approaches is conducted in the present section. The objectives of this study are summarized hereafter:

1. to evaluate the discrimination efficiency of the proposed methods, that is, their ability to extract small relevant subgraphs,
2. to observe the benefits of inflation (see section 8.6) with respect to the discrimination efficiency,
3. to measure practical run times and validate them against the theoretical complexity,
4. to evaluate how well the limited $\mathcal{K}$-walk approach can approximate the $\mathcal{K}$-walk approach for increasing $L_{max}$ values.

Randomly generated graphs are considered as well as two strongly connected components of the metabolic network representing the KEGG LIGAND database [55]. Random graphs were generated using the algorithm presented in [132]. This technique allows one to generate an undirected and unweighted graph with a prescribed degree sequence drawn from a power

law[9]. In our experiments, the exponent of the power law $\gamma$ is set to 2.5 and the $\mu$ parameter is tuned such that the mean degree equals 4, which are typical parameters used in [132]. Graphs were generated with sizes ranging from 100 to 20,000 nodes. The two strongly connected components extracted from KEGG contain respectively 7,695 and 18,297 nodes and their respective mean degrees are 2.7 and 2.9. Randomly generated graphs are undirected while the KEGG components are directed. Furthermore, we have defined weights on the graph edges such that $M_{qq'} = \frac{2}{d_q + d_{q'}}$ where $d_q$ and $d_{q'}$ are respectively the out degrees of nodes $q$ and $q'$. Hence any edge connected to a hub has a low conductance.

Each experiment is carried out using random sets of nodes of interest of size $|\mathcal{K}| = 2, 5, 10$ and 20. For each size, 10 sets are sampled out in order to produce averaged results with standard deviations.

### 8.8.1 Discrimination efficiency

The (limited) $\mathcal{K}$-walk approaches output relevance weights on the edges and nodes of the input graph. The total amount of information is defined as the sum of these relevances over all edges. Section 8.5 proposes several techniques to extract a relevant subgraph. In our experiments, a subgraph is extracted by keeping every edge with a relevance above a given threshold $\Theta_e$. The information captured by a subgraph is simply defined as the sum of the relevances over all edges in the subgraph. Dividing this quantity by the total information provides the *relative* captured information. Similarly, the subgraph relative size is obtained by dividing the number of edges in the subgraph by the number of edges in the input graph. Let us stress here that the selected performance measure, *i.e.* the captured information, is somewhat subjective as it directly relies on our node/edge relevance definitions (see definitions 43 and 44). Therefore, the results presented here might be thought of as an internal validation of our approaches. Besides, other related techniques use subjective or internal performance criteria as well. For instance, Newman [95] validates his approach qualitatively by discussing results obtained on social networks while Faloutsos et al. [48] use a performance measure based on the electrical current which is an internal measure used in their approach (see section 8.7). The performance of relevant subgraph extraction methods should ideally be assessed using an external criterion related to a real-world problem such as for instance the discovery of pathways in biological networks or the selection of an alternate itinerary on a roadmap in case of traffic jam.

[9]The probability that a node has a degree equal to $d$ is $P[d] = (d + \mu)^{-\gamma}$.

Figures 8.8 and 8.9 show the relative captured information for increasing relative subgraph sizes using the $\mathcal{K}$-walk method.

The $\mathcal{K}$-walk technique seems to offer a good discrimination efficiency since a large amount of information can be captured with small subgraphs. This is especially true when a small set of nodes of interest is considered. For instance, for an input graph with 1,000 nodes and $|\mathcal{K}| = 2$, a 10% subgraph captures in average 82% of information. In contrast, when $|\mathcal{K}| = 20$, it only captures 64% of information. The rationale is that a larger set of (randomly chosen) nodes of interest is more likely to cover many regions of the input graph, spreading the information in the graph. Furthermore, considering larger input graphs also reduces the covering of the nodes of interest, making the approach more discriminative.

We consider now the limited $\mathcal{K}$-walk approach. Figure 8.10 shows the relative discrimination efficiency for several $L_{max}$ values using an input graph with 1,000 nodes. This technique is more discriminative for small $L_{max}$ values. Indeed short walks only explore frequently a small portion of the input graphs, while concentrating the captured information. Note that using $L_{max} = 50$ provides almost the same results as the standard $\mathcal{K}$-walk approach (see the top of Figure 8.9). Using $L_{max}$ greater than 50 does not modify the discrimination efficiency. As for the $\mathcal{K}$-walk approach, a better discrimination is obtained with a smaller number of nodes of interest.

Next, we consider the KEGG directed network. The plot of captured information against extracted subgraph relative size using the $\mathcal{K}$-walk approach is presented on the top part of Figure 8.11. Relatively small portions of the input graph already contain a majority of the information of the graph. For instance, 10% of the input graph captures 67% of the total information. Note also that the captured information does not vary with the number of nodes of interest. We interpret this result as a consequence of the strong connectedness of the component extracted from the whole network. The information rapidly diffuses throughout such a graph and limited walks should reveal much better the influence of the specific nodes of interest chosen. The bottom part of Figure 8.11 illustrates that the captured information increases significantly faster with $L_{max} = 10$, with a slight dependence on $|\mathcal{K}|$. The behavior of the limited approach tends rapidly to the unlimited case for larger $L_{max}$ values. The same phenomenon is observed for the KEGG strongly connected component containing 18,297 nodes.

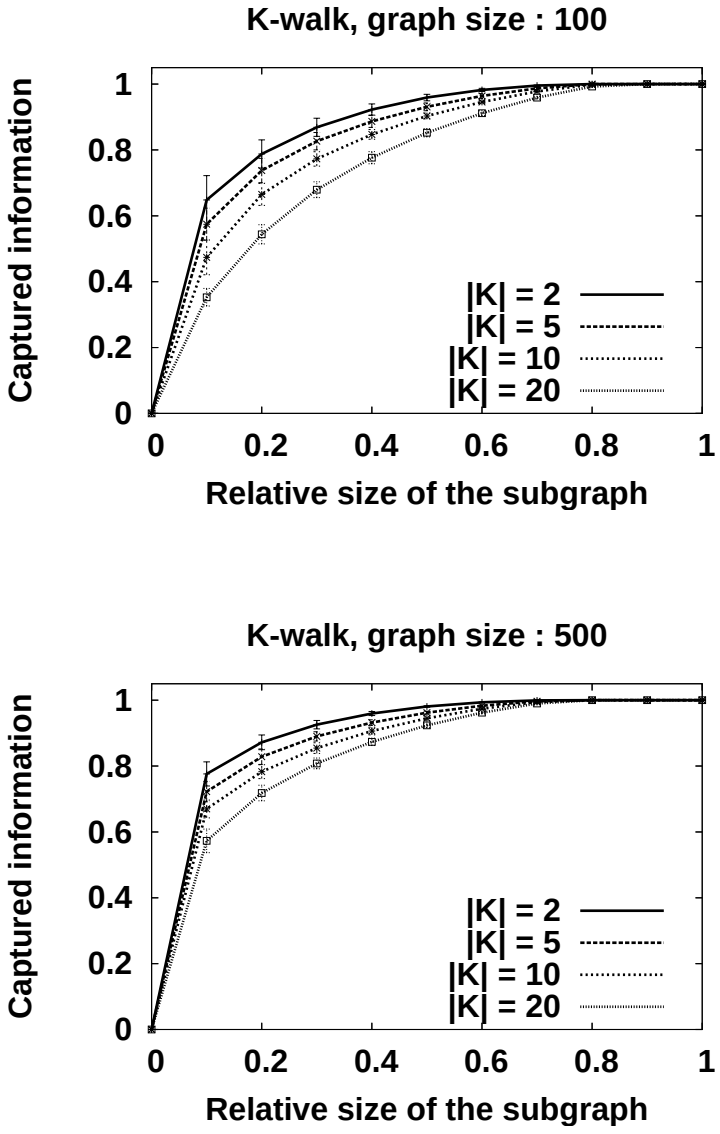



Figure 8.8: The relative captured information averaged over 10 random sets of nodes of interest for increasing relative subgraph sizes. Top: the $\mathcal{K}$-walk approach applied on a random input graph containing 100 nodes. Bottom: the $\mathcal{K}$-walk approach applied on a random input graph containing 500 nodes.

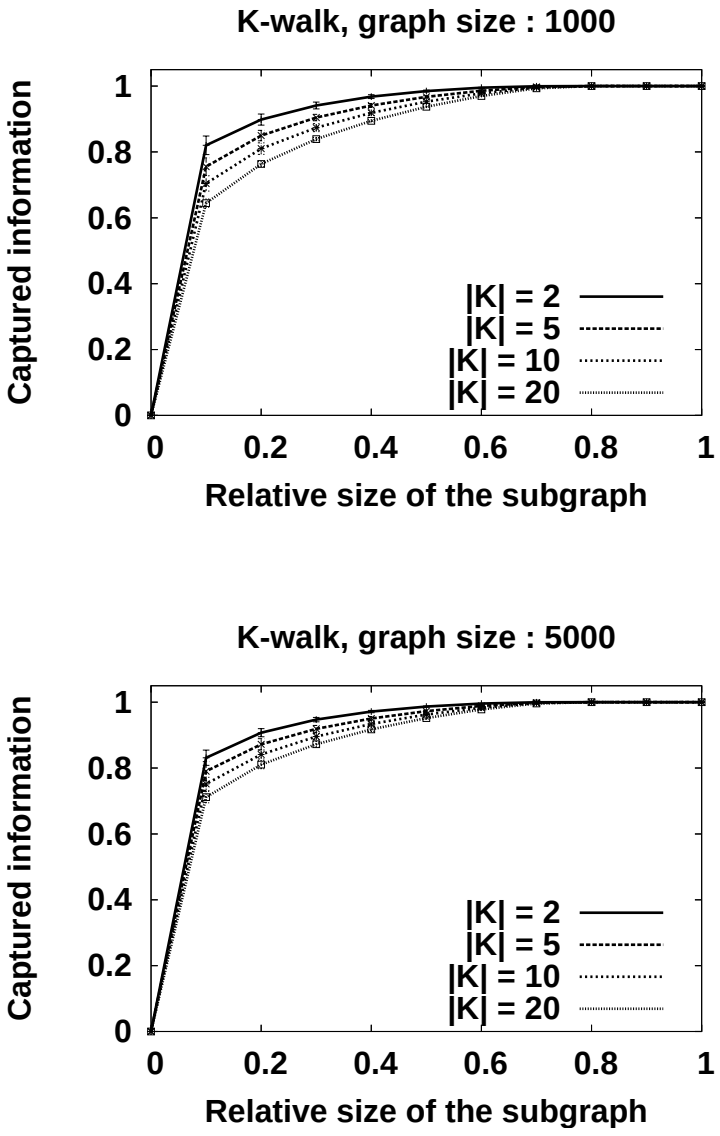



Figure 8.9: The relative captured information averaged over 10 random sets of nodes of interest for increasing relative subgraph sizes. Top: the $\mathcal{K}$-walk approach applied on a random input graph containing 1000 nodes. Bottom: the $\mathcal{K}$-walk approach applied on a random input graph containing 5000 nodes.

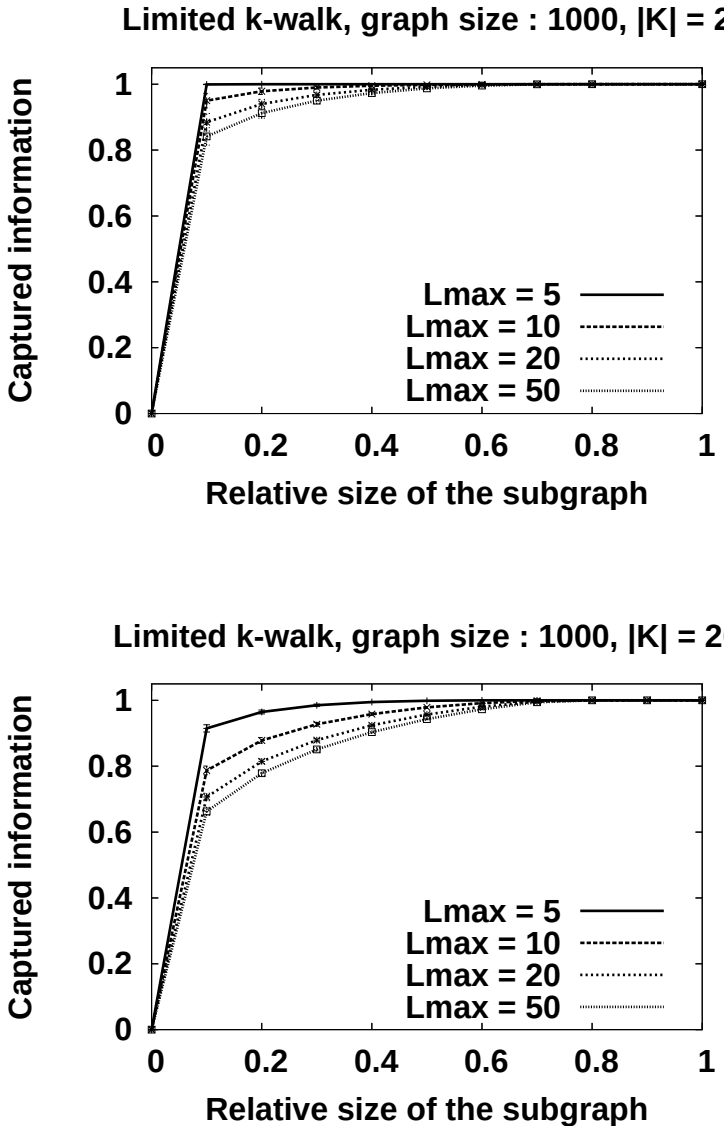



Figure 8.10: The relative captured information averaged over 10 samples of seed nodes for increasing $L_{max}$ values using the limited $\mathcal{K}$-walk approach.

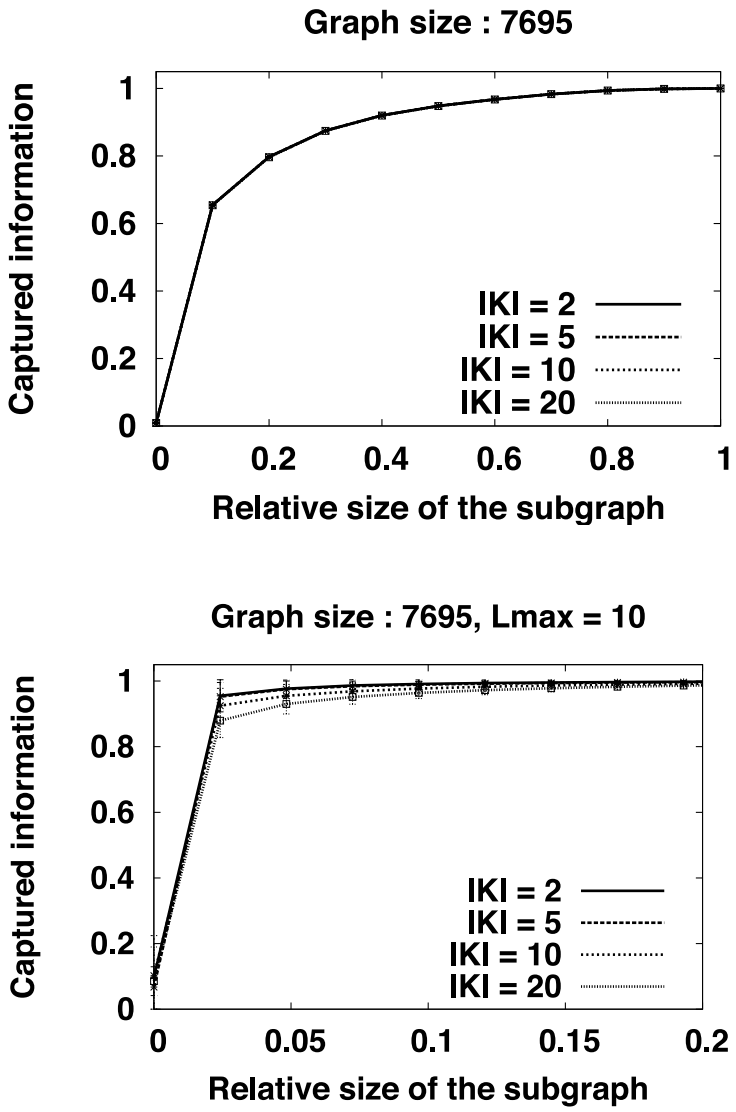



Figure 8.11: The relative captured information in a strongly connected component of the KEGG network containing 7,695 nodes. Top: The relative captured information using the $\mathcal{K}$-walk approach. Bottom: The relative captured information using the limited $\mathcal{K}$-walk approach with $L_{max} = 10$.

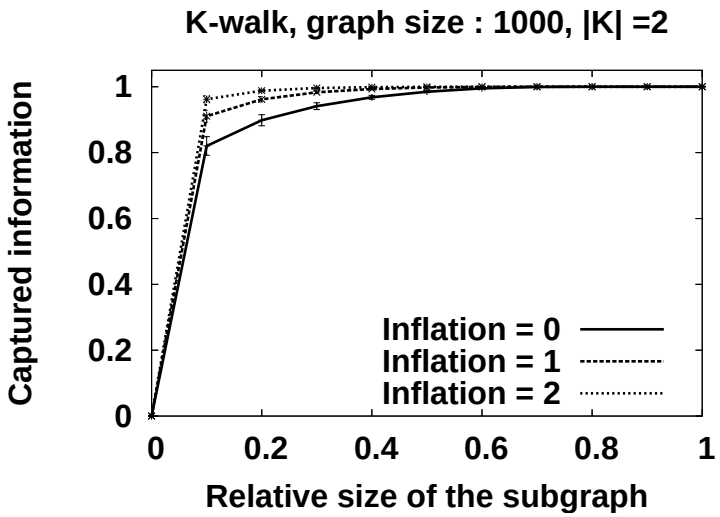



Figure 8.12: The influence of inflation on the discriminative efficiency using the $\mathcal{K}$-walk approach. Curves are provided using no inflation and using 1 and 2 inflation iterations.

### 8.8.2 Impact of inflation

Figure 8.12 shows the influence of inflation on the $\mathcal{K}$-walk approach using a random graph of size 1,000 and two nodes of interest. As expected, inflation allows one to improve the discriminative efficiency. For instance, a 10% subgraph captures in average 82% of information. Inflating the $\mathcal{K}$-walk approach once or twice captures respectively 91% and 96% of information with the same subgraph size. Similar observations are made when studying the impact of inflation on the limited $\mathcal{K}$-walk approach.

### 8.8.3 CPU time

The time complexity of the $\mathcal{K}$-walk approach is $O(|\mathcal{K}|n_s^3)$ (see section 8.2). We have assessed that our implementation fits this theoretical complexity. The top part of Figure 8.13 presents running times per node of interest for growing random graph sizes. The plotted times are computed by dividing the CPU time by the number of nodes of interest. Since the expected complexity is linear in $|\mathcal{K}|$, these normalized times should be cubic in $n_s$. The plot presented on the top of Figure 8.13 is a cubic curve fitting very well our experimental measures. The time complexity of the limited $\mathcal{K}$-walk approach is $\Theta(|\mathcal{K}|n_t L_{max})$. The bottom part of Figure 8.13 presents the CPU time normalized per walk step and number of nodes of interest. The bottom

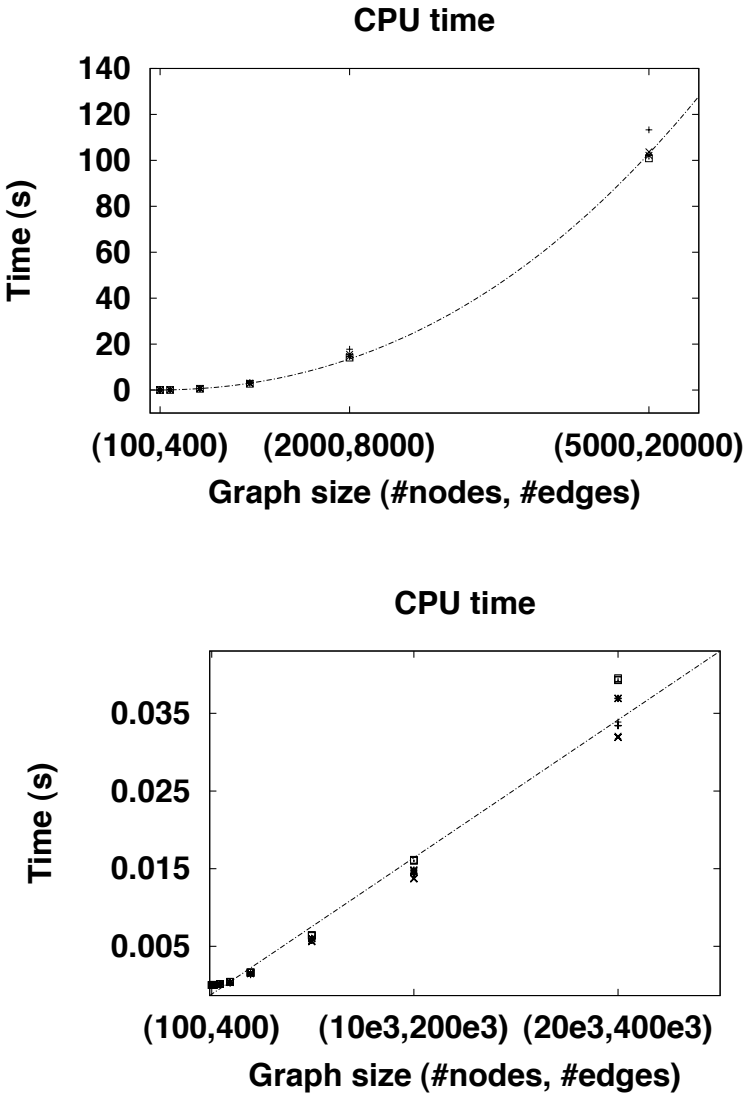



Figure 8.13: Top: running time results for the $\mathcal{K}$-walk approach. The running time divided by $|\mathcal{K}|$ is presented for various values of $|\mathcal{K}|$ and for increasing graph sizes. Bottom: Running time results for the limited $\mathcal{K}$-walk approach. The running time divided by $|\mathcal{K}|L_{max}$ is presented for various values of $|\mathcal{K}|$ and $L_{max}$ for increasing graph sizes.

of Figure 8.13 presents a least-square fit of a linear function to our experimental data. In a nutshell, the limited $\mathcal{K}$-walk approach can handle an input graph of 100,000 nodes, with 10 nodes of interest and $L_{max} = 1,000$ in about 30 minutes on a standard PC. The actual useful maximal length to consider is often significantly smaller as discussed in the next section.

### 8.8.4 Approximating $\mathcal{K}$-walks with limited $\mathcal{K}$-walks

As mentioned in the previous paragraph, the limited $\mathcal{K}$-walk offers a significant reduction of CPU time as compared to the $\mathcal{K}$-walk approach. Such a reduction allows one to handle much larger graphs. We study now whether the limited $\mathcal{K}$-walk approach can approximate adequately the subgraph extracted by the $\mathcal{K}$-walk approach. The limited approach is guaranteed to become equivalent to the unlimited case when $L_{max}$ tends to infinity (see section 8.3). We shall see that a good approximation is already obtained in practice with relatively small $L_{max}$ values.

A first indicator of the quality of the approximation is the cumulated absorption probability (see definition 17):

$$\frac{1}{|\mathcal{K}|} \sum_{x \in \mathcal{K}} \sum_{L=1}^{L_{max}} P[{}^{x}\mathcal{W}_L]$$

For a given starting node $x$, the inner sum computes the cumulative probability of walks up to length $L_{max}$. When all walk lengths are considered (i.e. $L_{max} \to \infty$), this quantity sums up to one. This cumulative probability is averaged over all starting nodes in $\mathcal{K}$. Hence, if this average is close to one for a finite $L_{max}$ value, the limited $\mathcal{K}$-walk is expected to approximate adequately the $\mathcal{K}$-walk approach.

Figure 8.14 displays the cumulated absorption probability with respect to the $L_{max}$ value for two input graphs containing respectively 5,000 and 20,000 nodes. A higher mass of absorption probability is obtained with a larger number of nodes of interest. For instance, for an input graph containing 20,000 nodes, an absorption probability of 0.91 is obtained with $L_{max} = 5,000$ and 20 nodes of interest while this probability mass is only 0.12 for 2 nodes of interest. The rationale is that the distance, in terms of number of steps, between two distinct nodes of interest becomes smaller in average with a larger number of nodes of interest. Therefore, smaller walk lengths are sufficient to get a high absorption probability mass.

The above analysis shows that only a small fraction of the total probability mass may be captured even for relatively large $L_{max}$ values. The extracted subgraph is however not necessarily very different with respect to the unlimited case. Indeed, *relative* edge relevances may be already well

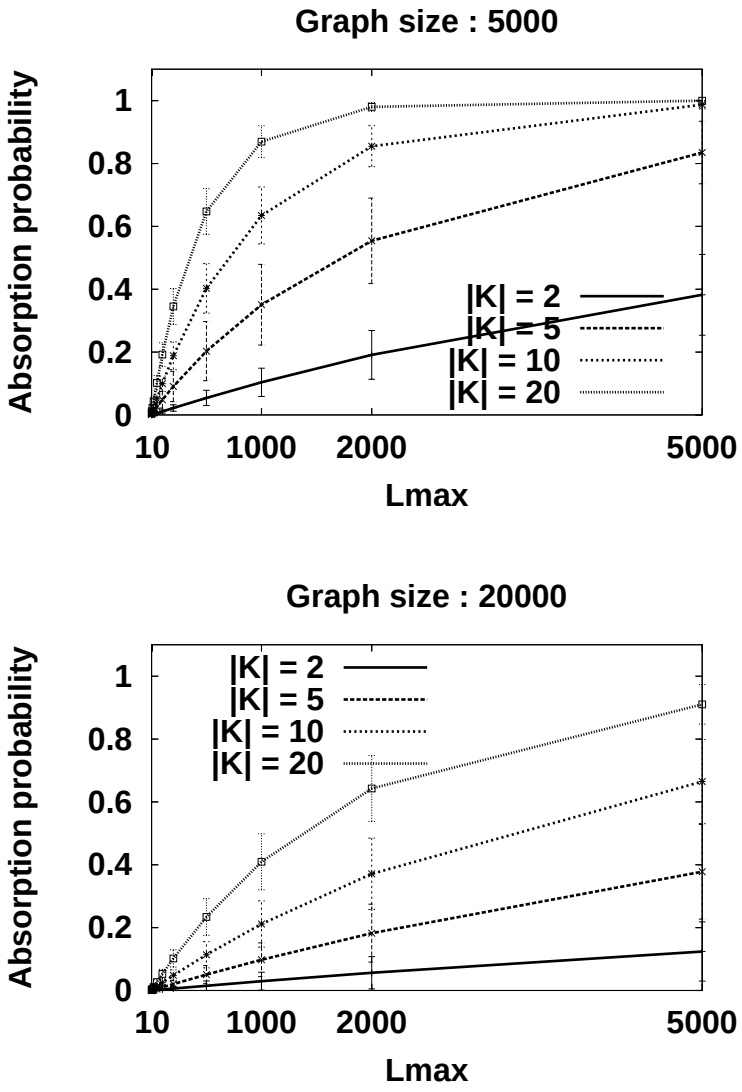



Figure 8.14: The cumulated absorption probability for increasing $L_{max}$ values, using the limited $\mathcal{K}$-walk approach.

approximated with a small $L_{max}$. The following experiment confirms this statement.

Several subgraphs are extracted with the $\mathcal{K}$-walk approach. Each subgraph corresponds to a fixed proportion (5%, 10% and 20%) of the same input graph. They serve as reference subgraphs. Next, the limited $\mathcal{K}$-walk approach is used to extract subgraphs. Figure 8.15 reports precision/recall figures of the edges included in these subgraphs with respect to the references. The curves are obtained while varying the selection threshold of the edge relevances computed with the limited approach. The plots correspond to several $L_{max}$ values with an input graph of 5,000 nodes and 5 nodes of interest. Unsurprisingly, the higher the $L_{max}$ value, the better the precision/recall. A very good precision/recall curve is already obtained using $L_{max} = 50$. This curve is almost ideal while the cumulated absorption probability is only equal to 0.2 in this case. In conclusion, the standard $\mathcal{K}$-walk approach can be approximated accurately with the limited $\mathcal{K}$-walk approach using a small $L_{max}$ value and at a much lower computational cost.

### 8.8.5 Implementation issues

The computation of the standard $\mathcal{K}$-walk approach essentially relies on matrix inversions. For this reason, we have chosen to implement a prototype in the Matlab 7.1 framework. The inversion routine proposed in Matlab allows us to deal with graphs containing about 5,000 nodes in a reasonable time. In practice however, we found out that the $\mathcal{K}$-walk approach could be more efficiently computed using the limited $\mathcal{K}$-walk approximation (see section 8.3.2). The latter approach has been implemented in the ANSI C programming language which is by far more efficient than Matlab when considering iterative procedures (*i.e.* the forward and backward recurrence computations, see section 8.3.3). It relies on sparse matrix representations, developed by our own, allowing one to deal with large sparse graphs containing for instance about 100,000 nodes. The implementations of the standard and limited $K$-walk approaches are available on the UCL Machine Learning Group (MLG) website: `http://www.ucl.ac.be/mlg/`.

## 8.9 Summary and discussion

We have presented novel methods to extract a relevant subgraph that best captures the relationships between $k$ given nodes of interest in a connected graph. The number $k$ of nodes of interest is typically assumed much smaller than the graph order $n$. The proposed approaches are based on random

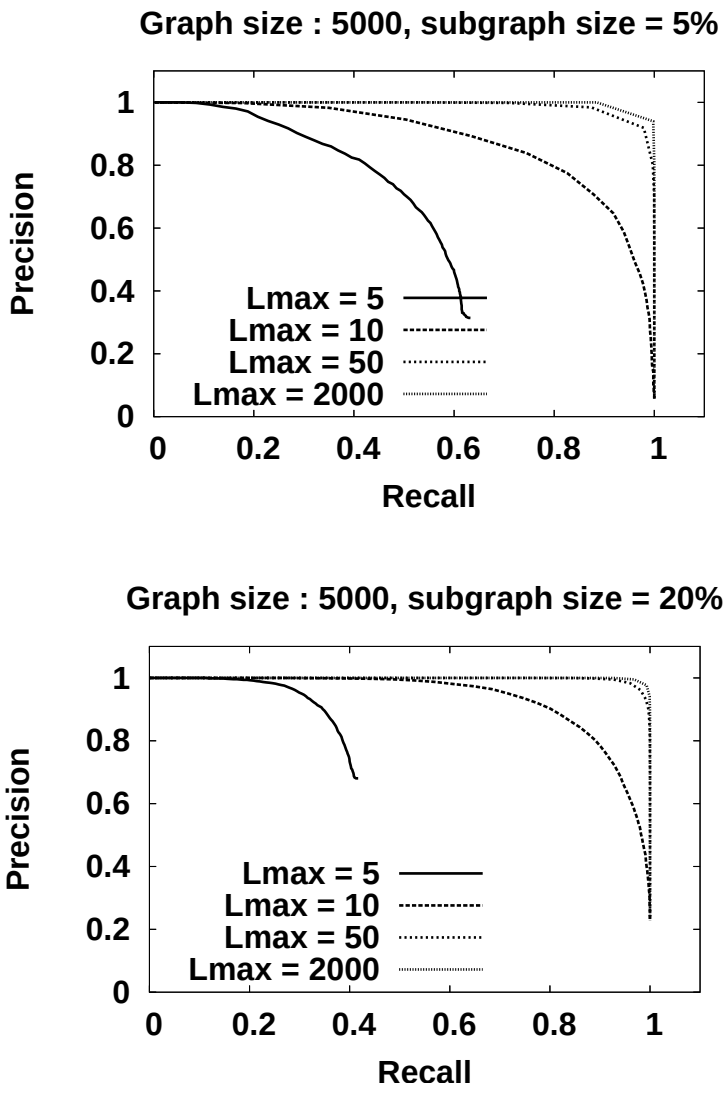



Figure 8.15: Measure of the quality of the approximation of the $\mathcal{K}$-walk approach by the limited $\mathcal{K}$-walk approach by comparing the extracted subgraphs. Subgraphs of relative sizes 5% (top) and 20% (bottom) are considered for the $\mathcal{K}$-walk approach. Precision/recall curves are computed by increasing the threshold on the edge relevance weight obtained with the limited $\mathcal{K}$-walk.

walks between the nodes of interest to define node and edge relevances proportionally to their expected frequency of use along these walks. Such quantities can be computed efficiently after transforming the graph successively into $k$ absorbing Markov chains. In particular, the time complexity of the $\mathcal{K}$-walk approach is $\mathcal{O}(kn_s^3)$ since it essentially amounts to invert $k$ matrices of size $\mathcal{O}(n_s) \times \mathcal{O}(n_s)$. The $\mathcal{K}$-walk approach relies on walks of any length between the nodes of interest. In contrast, a method relying only on shortest paths between these nodes would offer a much more restrictive view. The limited $\mathcal{K}$-walk approach offers an intermediate view (in number of walk steps) since it considers random walks up to a maximal length $L_{max}$. It can be efficiently implemented with two recurrences similar to those used in the expectation steps of the PHIT (see section 7.3.1) and POMMPHIT (see section 5.5.2) algorithms, respectively used in sequence classification and in Partially Observable Markov Models (POMMs) induction. The resulting time complexity is $\Theta(kn_tL_{max})$, where $n_t$ denotes the number of graph edges.

The limited $\mathcal{K}$-walk method offers an efficient way to approximate unlimited $\mathcal{K}$-walks even when a relatively small $L_{max}$ is chosen. The quality of this approximation is discussed in section 8.8. Practical experiments also illustrate that the proposed methods are well suited to extract relatively small subgraphs while capturing most of the information between the nodes of interest. Even more discriminative results can be obtained while inflating edge relevances. The limited $\mathcal{K}$-walk approach is clearly superior to the unlimited $\mathcal{K}$-walk with respect to CPU time. It can process a graph of 100,000 nodes with 10 nodes of interest and a maximal walk length of 1,000 steps in 30 minutes on a standard PC. Relevant subgraphs of the largest connected component (18,297 nodes and 53,248 edges) of the KEGG metabolic network [55] can be extracted in 3 seconds, while considering 5 randomly chosen nodes of interest and $L_{max} = 50$. This walk length is also shown to approximate very well the unlimited case. Additional variants of the proposed methods consider clusters of nodes of interest. Our future work includes several aspects described below.

Some of the practical experiments described in section 8.8 were conducted on metabolic networks. These experiments answer questions related to the amount of captured information in the extracted subgraphs or the respective results of the limited versus unlimited $\mathcal{K}$-walk methods. Average results were reported here with nodes of interest chosen at random. From a biological viewpoint the nodes of interest are far from random though. The proposed methods could be tested with carefully chosen nodes of interest. The extracted subgraphs could then be compared with known metabolic pathways. Interestingly, these pathways are not necessarily restricted to

single paths but several alternatives are often considered to define a biologically relevant subgraph. The initial edge weights should be defined in a meaningful way in this context, possibly according to the enthalpy of the various reactions represented by the nodes of the network. Such an approach would then help to predict relevant subgraphs which are not yet identified as known pathways.

The limited $\mathcal{K}$-walk approach is efficient since it scales linearly with $k$, the graph size and the maximal walk length. Yet it may require to be adapted for very large graphs by decomposing the computation over several predefined subgraphs and by combining the results. One could investigate whether some edges with very high relevances can be rapidly detected and serve as seeds of this process. Another open issue is the sensitivity of the $\mathcal{K}$-walk approach to the insertion or removal of an edge in the graph or, more finely, to some modification of its weight. Fast computation of the extracted subgraph after such a modification is a related issue. Up to now, the nodes of interest in the original graph are assumed to be given. The sensitivity analysis mentioned above could determine for which nodes of interest the results are more or less stable. A different but related problem is to find which nodes of interest have the highest influence overall in the graph. Some existing works precisely address this issue, motivated by the design of viral marketing strategies in social networks [109]. Finding the most influential nodes, according to standard models of information diffusion in social networks, is known to be a NP-hard optimization problem. However an efficient greedy strategy providing a good approximate solution has been proposed [74]. It would be interesting to study whether the $\mathcal{K}$-walk approach could be adapted to this objective.

Another interesting future research is concerned with inflation applied to limited $K$-walk up to a length $L_{\max}$. In a few words, inflation applies successively the $\mathcal{K}$-walk approach such that the edge relevances computed at iteration $i$ are used as the weight of the input graph at iteration $i+1$. It was shown that inflation increases the relevance contrast to better discriminate relevant/irrelevant parts of the graph. Inflation actually defines an iterative procedure closely related to the EM algorithm. Indeed, computing edge relevances provides the expectations of passage times on the edges of the graph. Besides, at the beginning of each inflation iteration, the edge weights are normalized to define a proper MC (see section 8.2.1). These two steps respectively matches the Expectation and Maximization steps of an EM algorithm (see for instance PHIT in section 7.3.1). We believe that this instance of EM would compute the transition probabilities maximizing the likelihood of the prescribed walk lengths (equivalent to first passage times)

$1, \ldots, L_{max}$ between the nodes of interest. The formal definition of the likelihood function as well as the convergence of inflation remain to be carefully investigated.

Information diffusion in graphs has also been studied with kernel methods [76]. Such a kernel defines a similarity measure between graph nodes. This kernel is related to a *lazy* random walk model. The random walker in this model has a certain probability, at each time step, to stay in place (even if the graph does not include self loops) rather than pursuing his walk. One could possibly extend this approach by defining a similarity measure between graph nodes which would become relative to some nodes of interest. A diffusion kernel is also strongly related to the graph Laplacian which plays a central role in spectral graph theory [32]. In particular, Shi and Malik introduced a *normalized cut* criterion to define an optimal bipartitioning of a graph [119]. They proposed a spectral segmentation approach based on the normalized Laplacian to approximate this problem. Meila and Shi also present links between this spectral segmentation and random walks in a Markov chain [90]. Random walks are also used in the MCL algorithm for graph clustering [128]. Another line of work would extend these works by considering an optimal partitioning or clustering of a graph relative to some predefined nodes of interest.

# Conclusion and future research

This thesis is concerned with the modeling of sequential processes from data. Such processes are here modeled using Markov models, namely finite order Markov chains (MC) and Hidden Markov Models (HMMs). Traditional approaches aim at estimating such models for predicting the next outcomes of a process given its past. Throughout this thesis we have adopted another point of view on modeling. Rather than focusing on *which* events will occur next, we have concentrated on the temporal dynamics of processes to predict *when* events occur. More precisely, the First Passage Times (FPT) are the times taken by the process between the realization of two considered events. These features are concerned with the global dynamics of the process, *i.e.* they are not restricted to a time-bounded window of events. The FPT dynamics of a process consists of the distributions of the FPT between any pairs of events occurring in the process. The *goal of this thesis* is to apply this process characterization to three distinct problems: (i) the induction of HMMs from data, (ii) the classification of discrete sequences and (iii) the mining of relevant subgraphs in large networks. The *results* obtained in this thesis are summarized hereafter.

### HMM induction

We have proposed a novel induction algorithm, called POMMStruct, to learn the structure and the parameters of Partially Observable Markov Models (POMMs). POMMs are HMMs constrained such that each state can only emit a single letter. We have shown that POMMs are as general as HMMs in the sense that they generate the same class of distributions on strings. POMMs are used as the target formalism since the FPT are more conveniently computed in these models while they preserve the expressiveness of HMMs. The state set of a POMM can be partitioned into *blocks* gathering states emitting the same letter. The FPT between symbols in a POMM are concerned with the time taken by the process to go from

one block to another. We have shown that FPT in POMMs are of phase type (PH). POMMStruct aims at inducing a POMM to best reproduce the FPT dynamics observed in the learning sample. We have motivated the use of FPT in model induction by showing that they are informative about the structure to be learned. For instance, multi-modal FPT distributions reveal the presence of dominant path length between blocks in the model while the FPT directly measure the time between conditioning and conditioned events in long-term dependencies. Starting from a standard Markov chain (MC), POMMStruct iteratively adds states to the model. The probabilistic parameters are estimated using a novel method based on the EM algorithm, called POMMPHit. The latter estimates parameters to maximize the likelihood of the observed FPT. POMMPHit differs from the standard Baum-Welch procedure since the likelihood function to be maximized is concerned with times between events (*i.e.* emission of symbols) rather than by the complete generative process. Infrequently triggered transitions are trimmed off from the model according to a measure based on passage times. We have presented experimental results obtained with POMMStruct on several sequence modeling tasks. We have shown that POMMStruct is able to model a wide range of FPT dynamics including multi-modal distributions, shifted distributions and long-term dependencies. Comparative results with Baum-Welch applied on fully connected graphs with transition trimming and the Bayesian state merging approach due to Stolcke have shown that POMMStruct is better suited to fit a process with a complex FPT dynamics. On symbol prediction tasks, models obtained with POMMStruct and reestimated with Baum-Welch outperform the approach of Stolcke. POMMStruct does not always offer a better prediction performance than Baum-Welch alone but the FPT are more accurately modeled.

Limitations of the current approach and possible future researches in that perspective are now discussed. A first remark concerns the use of POMMs instead of HMMs in our induction technique. While being equivalent to HMMs, POMMs are often less concise than HMMs and therefore harder to interpret. POMMStruct could be adapted to apply to HMMs or one could apply a reduction algorithm, such as the one proposed in [7], to the estimated POMM. Another possible research would consider higher order events, that is modeling the FPT between occurrences of subsequences. For instance, it could be used to model long-term dependencies between aggregated events (*i.e.* subsequences). Finally, smoothing issues are an important problem for HMM or POMM induction as the learning samples are often very sparse. Traditional techniques smooth models with respect to the sequence likelihood. Alternatively, one could focus on smoothing the FPT.

That is, one wants to spread the probability mass to rare or unobserved FPT. Modeling FPT with PH distributions offers a natural smoothing controlled by the number of phases. The number of phases corresponds here to the number of states in POMMs. One could therefore control the number of state additions (and also the transition trimming) towards this smoothing objective.

### Sequence classification

We have proposed a novel approach to the discrete sequence classification problem. It relies on FPT features extracted from the sequences to classify. In contrast with POMMStruct, the FPT considered here are measured between occurrences of subsequences up to a prescribed length rather than between individual symbols. Due to the potentially large number of features (*i.e.* subsequence pairs), a feature selection procedure, based on the Jensen-Shannon divergence, has been proposed. The selected features are modeled with phase-type (PH) distributions which also characterize the FPT dynamics in HMMs and POMMs. Doing so, one includes a part of the powerful expressiveness of HMMs into the resulting classifier. The PHit algorithm, an adapted version of the Expectation-Maximization algorithm, has been proposed to estimate PH distributions from data. It can be thought of as a simpler version of the POMMPHit algorithm used to estimate the probabilistic parameters of POMMs. In contrast with this technique, each feature is fitted on a separate model being an absorbing MC. Tuning the number of phases deals with the bias-variance trade-off since using a small number of phases smooths the distribution while using many phases tends to overfit the observed FPT. The estimated PH distributions are used to build two classifiers: (i) a generative classifier computing the likelihood with PH distributions and (ii) a Support Vector Machine (SVM) classifier applied in an embedding based on PH distributions. Comparative results on DNA splicing site detection and on protein sublocalization tasks have shown that our approach compares favorably with states-of-the art techniques relying on local features.

We now discuss the limitations of the current work and possible extensions to be addressed in a future research. While the current feature selection procedure is efficient to select discriminative pairs, the selection may include some redundancy. This is particularly the case when features have overlapping subsequences, *e.g.* (`aba, baa`) and (`ba, ba`). A possible solution to this problem would filter out such overlapping features. A more refined strategy would combine the JS divergence with a criterion such as the mutual information [57] to trade off discriminative efficiency and min-

imal redundancy. Another perspective concerns the tuning of the number of phases. The complexity of the FPT dynamics may vary from one feature to another and the number of phases should be adapted accordingly. A more powerful approach would automatically discover the optimal structure, including the number of phases, of the absorbing MC associated to PH distributions. The POMMStruct algorithm (see section 5.5) used to learn the structure of POMMs could be simplified towards this objective.

### Subgraph mining

We have introduced a new method for mining relevant subgraphs in large networks. More precisely, given a graph and a set $\mathcal{K}$ of nodes of interest, the proposed technique extracts a subgraph modeling the relationships between the nodes of interest. The relevance criterion used to extract a subgraph is defined in a Markovian perspective. This criterion relies on all possible ways to connect the nodes of interest (each way having a certain likelihood) and not only shortest-distance or maximal-flow paths. The input graph is first transformed into a MC by normalizing the graph adjacency matrix into a stochastic transition matrix. Performing random walks on this MC defines a sequential process. A $\mathcal{K}$-walk is defined as a random walk starting in a node of interest and ending when any other node of interest is entered for the first time. Relevant nodes and edges are then defined as the most frequently visited in average during these walks. Alternatively, we have proposed to consider $\mathcal{K}$-walks of a limited length. Such a constraint incorporates a prior knowledge about the maximum path length allowed to explain the relation between the nodes of interest. An efficient algorithm has been proposed to compute node and edge relevances in the case of such limited walks. This technique can approximate the standard $\mathcal{K}$-walk approach by considering a large but finite maximum walk length. In practice, we have shown that moderate walk lengths already approximate the standard approach very well. Several strategies based on the computed relevances have been proposed to extract a subgraph. Such techniques greedily search for a maximally relevant subgraph of minimal size, possibly constrained to be connected. Additionally, an inflation procedure, consisting in applying our approaches recursively, has been proposed to enforce more discrimination for extracting smaller subgraphs. Experiments on the KEGG metabolic network [55] as well as on artificially generated graphs are reported. Our approach is shown to have a good discriminative efficiency, *i.e.* it can extract small graphs having a high relevance. From the computational viewpoint, relevant subgraphs of the largest connected component (18,297 nodes and 53,248 edges) of the KEGG metabolic network [55] can be extracted in 3 seconds on a standard PC, while considering 5 randomly chosen nodes of

interest and $\mathcal{K}$-walk up to length 50.

As a future research we propose to apply the $\mathcal{K}$-walk approach to biological applications such as the discovery of new pathways in metabolic networks. The initial edge weights of the input graph should be defined in a meaningful way in this context, possibly according to the enthalpy of the various reactions represented by the nodes of the network. Interestingly, pathways are not necessarily restricted to single paths but several alternatives are often considered, hence searching for a subgraph makes sense in such a context. To assess the performance of our method one could select a few bioentities which are parts of a known pathway and compare the extracted subgraph with the complete considered pathway. Another future research is concerned with inflation applied to limited $\mathcal{K}$-walk. In a few words, inflation defines an iterative procedure closely related to the EM algorithm. We believe that this instance of EM would compute the transition probabilities maximizing the likelihood of the prescribed walk lengths (equivalent to first passage times) between the nodes of interest. The formal definition of the likelihood function being maximized as well as the convergence of inflation remain to be carefully investigated. Another possible future work is concerned with a sensitivity analysis of the relevance measure to graph modifications such as edge reweighting or deletion. The present work could also be an inspiring approach to define new kernels on graph for computing node similarities relatively to a set of nodes of interest.

To sum up, the FPT dynamics in Markov models has been used in turn to solve three distinct problems, namely HMM induction, sequence classification and graph mining. We have shown that these features provide an interesting *fitting criterion* to discover the structure of POMMs, they also contains much *discriminative information* to classify sequences and finally they can be used to define a *relevance measure* for mining subgraphs relatively to a set of prescribed nodes of interest.

# Bibliography


[1] N. Abe and M. Warmuth. On the computational complexity of approximating distributions by probabilistic automata. *Machine Learning*, 9:205–260, 1992.

[2] H. Akaike. A new look at the statistical identification model. *IEEE Transactions on Automatic Control*, 19:716–723, 1974.

[3] Yasemin Altun, Ioannis Tsochantaridis, and Thomas Hofmann. Hidden markov support vector machines. In *ICML*, pages 3–10, 2003.

[4] Erling D. Andersen, Cornelis Roos, and Tamás Terlaky. On implementing a primal-dual interior-point method for conic quadratic optimization. *Math. Program.*, 95(2):249–277, 2003.

[5] Søren Asmussen, Olle Nerman, and Marita Olsson. Fitting phase-type distributions via the em algorithm. *Scandinavian Journal of Statistics*, 23(4):419–441, 1996.

[6] L. R. Bahl, P. F. Brown, P. V. de Souza, and R. L. Mercer. Maximum mutual information estimation of hidden Markov model parameters for speech recognition. In *Proceedings IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 49–52, Tokyo, 1986.

[7] Vijay Balasubramanian. Equivalence and reduction of hidden markov models. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1993.

[8] L. Baum and T. Petry. Statistical inference for probabilistic functions of finite state markov chains. *Annals of Mathematical Statistics*, 37:1554–1563, 1966.

[9] R. Begleiter, R. El-Yaniv, and G. Yona. On prediction using variable order markov models. *Journal of Artificial Intelligence Research*, 22:385–421, 2004.

[10] Y. Bengio and P. Frasconi. Diffusion of context and credit information in markovian models. *Journal of Artificial Intelligence Research*, 3:223–244, 1995.

[11] Christopher M. Bishop. *Neural networks for pattern recognition*. Oxford University Press, Oxford, UK, 1996.

[12] D. Blackwell and L.Koopmans. On the identifiability problem for functions of finite markov chains. *Annals of Mathematical Statistics*, 28:1010–11015, 1957.

[13] A. Bobbio, A. Horváth, M. Scarpa, and M. Telek. Acyclic discrete phase type distributions: properties and a parameter estimation algorithm. *Perform. Eval.*, 54(1):1–32, 2003.

[14] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[15] M. Brand. A random walks perspective on maximizing satisfaction and profit. *Proceedings of the 2005 SIAM International Conference on Data Mining*, 2005.

[16] M.E. Brand. Structure learning in conditional probability models via an entropic prior and parameter extinction. *Neural Computation Journal*, 11(5):1155–1182, 1999.

[17] U. Brandes and T. Erlebach, editors. *Network Analysis: Methodological Foundations*. Number 3418 in Lecture Notes in Computer Science. Springer, 2005.

[18] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[19] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[20] J. Callut. Mémoire de fin d'études : Implémentation efficace des support vector machines pour la classification. Technical report, Université Libre de Bruxelles (ULB), 2003.

[21] J. Callut and P. Dupont. A Markovian approach to the induction of regular string distributions. In *Grammatical Inference: Algorithms and Applications*, number 3264 in Lecture Notes in Artificial Intelligence, pages 77–90, Athens, Greece, 2004. Springer Verlag.

[22] J. Callut and P. Dupont. F$\beta$ support vector machines. In *International Joint Conference on Neural Networks 2005, (IJCNN 2005)*, pages 1443–1448, Montreal, Canada, 2005. IEEE.

[23] J. Callut and P. Dupont. Inducing hidden markov models to model long-term dependencies. In *16th European Conference on Machine Learning (ECML)*, number 3720 in Lecture Notes in Artificial Intelligence, pages 513–521, Porto, Portugal, 2005. Springer Verlag.

[24] J. Callut and P. Dupont. Sequence discrimination using phase-type distributions. In *17th European Conference on Machine Learning (ECML)*, number 4212 in Lecture Notes in Artificial Intelligence, pages 78–89, Berlin, Germany, 2006. Springer Verlag.

[25] J. Callut and P. Dupont. Learning hidden markov models from first passage times. In *18th European Conference on Machine Learning (ECML)*, number 4701 in Lecture Notes in Artificial Intelligence, pages 91–103, Warsaw, Poland, 2007. Springer Verlag.

[26] R. Carrasco and J. Oncina. Learning stochastic regular grammars by means of a state merging method. In *Grammatical Inference and Applications, ICGI'94*, number 862 in Lecture Notes in Artificial Intelligence, pages 139–150, Alicante, Spain, 1994. Springer Verlag.

[27] R. Carrasco and J. Oncina. Learning deterministic regular gramars from stochastic samples in polynomial time. *Theoretical Informatics and Applications*, 33(1):1–19, 1999.

[28] F. Casacuberta. Some relations among stochastic finite state networks used in automatic speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):691–695, 1990.

[29] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001.

[30] O. Chapelle. Training a support vector machine in the primal. Technical Report 147, MPI, 04 2006.

[31] Olivier Chapelle, Vladimir Vapnik, Olivier Bousquet, and Sayan Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1-3):131–159, 2002.

[32] F.R. Chung. *Spectral graph theory.* American Mathematical Society, 1997.

[33] Alexander Clark and Franck Thollard. Pac-learnability of probabilistic deterministic finite state automata. *J. Mach. Learn. Res.*, 5:473–497, 2004.

[34] Michael Collins. Discriminative training methods for hidden markov models: theory and experiments with perceptron algorithms. In *EMNLP '02: Proceedings of the ACL-02 conference on Empirical methods in natural language processing*, pages 1–8, Morristown, NJ, USA, 2002. Association for Computational Linguistics.

[35] R. M. Corless, G. H. Gonnet, D. E. G. Hare, D. J. Jeffrey, and D. E. Knuth. On the Lambert $W$ function. *Adv. Comput. Math.*, 5(4):329–359, 1996.

[36] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. Royal Statistical Society Ser. B (methodological)*, 39:1–38, 1977.

[37] F. Denis and Y. Esposito. Learning classes of probabilistic automata. In *Proc. of 17th Annual Conference on Learning Theory (COLT)*, number 3120 in Lecture Notes in Computer Science, pages 124–139, Banff, Canada, 2004. Springer Verlag.

[38] Yves Deville, David Gilbert, Jacques van Helden, and Shoshana J. Wodak. An overview of data models for the analysis of biochemical pathways. *Briefings in Bioinformatics*, 4(3):246–259, 2003.

[39] Chris Ding, Rong Jin, Tao Li, and Horst Simon. A learning framework using green's function and kernel regularization with application to recommender system. In *To appear in the proceedings of the international conference on Knowledge Discovery in Databases (KDD2007)*, 2007.

[40] P. Doyle and J. Snell. *Random walks and electric networks*. The Mathematical Association of America, 1984.

[41] P. Dupont. Noisy sequence classification with smoothed markov chains. In *Conférence francophone sur l'apprentissage automatique 2006, (CAp 2006)*, pages 187–201, Trégastel, France, 2006.

[42] P. Dupont and J.C. Amengual. Smoothing probabilistic automata: an error-correcting approach. In A. Oliveira, editor, *Grammatical Inference: Algorithms and Applications*, number 1891 in Lecture Notes in Artificial Intelligence, pages 51–64, Lisbon, Portugal, 2000. Springer Verlag.

[43] P. Dupont, J. Callut, G. Dooms, J.-N. Monette, and Y. Deville. Relevant subgraph extraction from random walks in a graph. Technical Report RR 2006-07, INGI, UCL, 2006.

[44] P. Dupont and L. Chase. Using symbol clustering to improve probabilistic automaton inference. In *Grammatical Inference, ICGI'98*, number 1433 in Lecture Notes in Artificial Intelligence, pages 232–243, Ames, Iowa, 1998. Springer Verlag.

[45] P. Dupont, F. Denis, and Y. Esposito. Links between Probabilistic Automata and Hidden Markov Models: probability distributions, learning models and induction algorithms. *Pattern Recognition*, 38(9):1349–1371, 2005.

[46] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis*. Cambridge University Press, 1998.

[47] D. Eppstein. Finding the $k$ shortest paths. *SIAM Journal on Computing*, 28(2):652–673, 1999.

[48] C. Faloutsos, K. McCurley, and A. Tomkins. Fast discovery of connection subgraphs. In *10th ACM Conference on Knowledge Discovery and Data Mining (KDD)*, volume 2, pages 118–127, August 2004.

[49] G.D. Forney. The Viterbi algorithm. *IEEE Proceedings*, 3:268–278, 1973.

[50] F. Fouss, A. Pirotte, J.-M. Renders, and M. Saerens. Random-walk computation of similarities between nodes of a graph, with application to collaborative recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 19(3):355–369, 2007.

[51] L.C. Freeman. A set of measures of centrality based upon betweenness. *Sociometry*, 40:35–41, 1977.

[52] D. Freitag and A. McCallum. Information extraction with HMM structures learned by stochastic optimization. In *Proc. of the Seventeenth National Conference on Artificial Intelligence, AAAI*, pages 584–589, 2000.

[53] M.R. Garey and D.S. Johnson. *Computers and Intractability: A guide to the theory of NP-Completenes*. Freeman and Company, San Francisco, 1979.

[54] P. S. Gopalakrishnan, Dimitri Kanevsky, Arthur Nádas, and David Nahamoo. An inequality for rational functions with applications to some statistical estimation problems. *IEEE Transactions on Information Theory*, 37(1):107–113, 1991.

[55] S. Goto, T. Nishioka, and M. Kanehisa. Ligand: chemical database of enzyme reactions. *Nucleic Acids Research*, 28(1):380–382, 2000.

[56] Amaury Habrard, François Denis, and Yann Esposito. Using pseudo-stochastic rational languages in probabilistic grammatical inference. In *ICGI*, pages 112–124, 2006.

[57] M. Hall. Correlation-based feature selection for machine learning, 1998.

[58] Jihun Ham, Daniel Lee, Sebastian Mika, and Bernhard Scholkopf. A kernel view of the dimensionality reduction of manifolds. *Proceedings of the 21st International Conference on Machine Learning (ICML2004)*, 2004.

[59] David Harel and Yehuda Koren. On clustering using random walks. *Proceedings of the conference on the Foundations of Software Technology and Theoretical Computer Science; Lecture Notes in Computer Science*, 2245:18–41, 2001.

[60] Trevor Hastie, Saharon Rosset, Robert Tibshirani, and Ji Zhu. The entire regularization path for the support vector machine. *J. Mach. Learn. Res.*, 5:1391–1415, 2004.

[61] Xiaodong He and Li Deng. A new look at discriminative training for hidden markov models. *Pattern Recognition Letters*, 28(11):1285–1294, 2007.

[62] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[63] F.K. Hwang, D.S. Richards, and P. Winter. *The Steiner Tree Problem*, volume 53 of *Annals of Discrete Mathematics*. Elsevier, North-Holland, 1992.

[64] Tommi S. Jaakkola and David Haussler. Exploiting generative models in discriminative classifiers. In *Advances in neural information processing systems 11*, pages 487–493, Cambridge, MA, USA, 1998. MIT Press.

[65] Tony Jebara and Alex Pentland. Maximum conditional likelihood via bound maximization and the cem algorithm. In *Proceedings of the 1998 conference on Advances in neural information processing systems II*, pages 494–500, Cambridge, MA, USA, 1999. MIT Press.

[66] F. Jelinek. *Statistical Methods for Speech Recognition*. MIT Press, Cambridge, MA, 1998.

[67] Victor M. Jiménez and Andrés Marzal. Computing the k shortest paths: A new algorithm and an experimental comparison. In *Algorithm Engineering: 3rd International Workshop, WAE'99*, volume 1668, pages 15–29, London, UK, 1999. Springer.

[68] T. Joachims. *Learning to Classify Text Using Support Vector Machines – Methods, Theory, and Algorithms*. Kluwer/Springer, 2002.

[69] Thorsten Joachims. Making large-scale support vector machine learning practical. *Advances in Kernel Methods*, pages 169–184, 1999.

[70] B.-H. Juang, W. Chou, and C.-H. Lee. Minimum Classification Error Rate Methods for Speech Recognition. *IEEE Transactions on Speech and Audio Processing*, 5(3):257–265, 1997.

[71] W. Karush. Minima of functions of several variables with inequalities as side constraints. Master's thesis, Dept. of Mathematics, Univ. of Chicago, 1939.

[72] S.M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustic, Speech and Signal Processing*, 35(3):400–401, 1987.

[73] J.G. Kemeny and J.L. Snell. *Finite Markov Chains*. Springer-Verlag, 1983.

[74] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence trhough a social network. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 137–146, Washington, DC, USA, 2003.

[75] R. Kneser and H. Ney. Improved backing-off for m-gram language modeling. In *International Conference on Acoustic, Speech and Signal Processing*, pages 181–184, Detroit, Michigan, 1995.

[76] R. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete input spaces. In *Proc. of the Nineteenth International Conference on Machine Learning*, pages 316–322, 2002.

[77] H. W. Kuhn and A. W. Tucker. Nonlinear programming. In *Proc. $2^{nd}$ Berkeley Symposium on Mathematical Statistics and Probabilistics*, pages 481–492, Berkeley, 1951. University of California Press.

[78] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML '01: Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[79] Gert R. G. Lanckriet, Nello Cristianini, Peter L. Bartlett, Laurent El Ghaoui, and Michael I. Jordan. Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research*, 5:27–72, 2004.

[80] G. Latouche and V. Ramaswami. *Introduction to Matrix Analytic Methods in Stochastic Modeling*. Society for Industrial & Applied Mathematics,U.S., 1999.

[81] Christina S. Leslie, Eleazar Eskin, Adiel Cohen, Jason Weston, and William Stafford Noble. Mismatch string kernels for discriminative protein classification. *Bioinformatics*, 20(4):467–476, 2004.

[82] Christina S. Leslie, Eleazar Eskin, and William Stafford Noble. The spectrum kernel: A string kernel for svm protein classification. In *Pacific Symposium on Biocomputing*, pages 566–575, 2002.

[83] E. Levin and R. Pieraccini. Planar Hidden Markov modeling: from speech to optical character recognition. In C.L. Giles, S.J. Hanton, and J.D. Cowan, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 731–738. Morgan Kauffman, 1993.

[84] Jie Li, Jiaxin Wang, Yannan Zhao, and Zehong Yang. Self-adaptive design of hidden markov models. *Pattern Recogn. Lett.*, 25(2):197–210, 2004.

[85] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American Society for Information Science and Technology*, 58(7):1019–1031, 2007.

[86] J. Lin. Divergence measures based on the shannon entropy. *IEEE Trans. Information Theory*, 37:145–151, 1991.

[87] D. Llorens, J.-M. Vilar, and F. Casacuberta. Finite state language models smoothed using n-grams. *International Journal of Pattern Recognition and Artificial Intelligence*, 16(3):275–289, 2002.

[88] Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444, 2002.

[89] Andrew McCallum, Dayne Freitag, and Fernando C. N. Pereira. Maximum entropy markov models for information extraction and segmentation. In *ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning*, pages 591–598, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[90] M. Meila and J. Shi. A random walks view of spectral segmentation. In *Proc. of AISTATS*, 2001.

[91] J. Mercer. Functions of positive and negative type and their connection with the theory of integral equations. *Philos. Trans. Roy. Soc. London*, A 209:415–446, 1909.

[92] Carl D. Meyer. *Matrix analysis and applied linear algebra*. Society for Industrial and Applied Mathematics, 2000.

[93] John Shawe-Taylor Nello Cristianini. *An Introduction to Support Vector Machines*. The Press Syndicate of the University of Cambridge, 2002.

[94] M. F. Neuts. *Matrix-Geometric Solutions in Stochastic Models*. John Hopkins, University Press, Baltimore, 1981.

[95] M.E.J. Newman. A measure of betweenness centrality based on random walks. *Social networks*, 27:39–54, 2005.

[96] H. Ney, U. Essen, and R. Kneser. On structuring probabilistic dependences in stochastic language modelling. *Computer Speech and Language*, 8:1–38, 1994.

[97] Mordechai Nisenson, Ido Yariv, Ran El-Yaniv, and Ron Meir. Towards behaviometric security systems: Learning to identify a typist. In *PKDD*, pages 363–374, 2003.

[98] J. R. Norris. *Markov Chains*. Cambridge University Press, United Kingdom, 1997.

[99] M. Ostendorf and H. Singer. HMM topology design using maximum likelihood successive state splitting. *Computer Speech and Language*, 11:17–41, 1997.

[100] E. Osuna, R. Freund, and F. Girosi. Improved training algorithm for support vector machines, 1997.

[101] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. *Technical Report 1999-0120, Computer Science Department, Stanford University*, 1999.

[102] J. C. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical Report MSR-TR-98-14, Microsoft Research, 1998.

[103] A. Poritz. Hidden markov models: a guided tour. In *International Conference on Acoustic, Speech and Signal Processing*, pages 7–13, 1988.

[104] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 1992.

[105] Huaijun Qiu and Edwin R. Hancock. Image segmentation using commute times. *Proceedings of the 16th British Machine Vision Conference (BMVC 2005)*, pages 929–938, 2005.

[106] Huaijun Qiu and Edwin R. Hancock. Clustering and embedding using commute times. *To appear in the IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2007.

[107] L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[108] L. Rabiner and B.-H. Juang. *Fundamentals of Speech Recognition*. Prentice-Hall, 1993.

[109] M. Richardson and P. Domingos. Mining knowledge-sharing sites for viral marketing. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 61–70, 2002.

[110] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[111] Jonathan Le Roux and Erik McDermott. Optimization methods for discriminative training. In *Eurospeech*, pages pp. 3341–3344, 2005.

[112] Marco Saerens, Francois Fouss, Luh Yen, and Pierre Dupont. The principal components analysis of a graph, and its relationships to spectral clustering. *Proceedings of the 15th European Conference on Machine Learning (ECML 2004). Lecture Notes in Artificial Intelligence, vol. 3201, Springer-Verlag, Berlin*, pages 371–383, 2004.

[113] Jarkko Salojärvi, Kai Puolamäki, and Samuel Kaski. Expectation maximization algorithms for conditional likelihoods. In Luc De Raedt and Stefan Wrobel, editors, *Proceedings of the 22nd International Conference on Machine Learning*, pages 753–760, 2005.

[114] Craig Saunders, John Shawe-Taylor, and Alexei Vinokourov. String kernels, fisher kernels and finite state automata. In *NIPS*, pages 633–640, 2002.

[115] B. Schölkopf and A. Smola. *Learning with Kernels*. MIT Press, Cambridge, 2002.

[116] G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978.

[117] E. Senata. *Non-negative Matrices and Markov Chains*. Springer-Verlag, 1981.

[118] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, June 2004.

[119] J. Shi and J. Malik. Normalised cuts and image segmentation. *IEEE Transactions on Pattern Matching and Machine Intelligence*, 22:888–905, August 2000.

[120] B. Stenger, V. Ramesh, N. Paragios, F. Coetzee, and J. Buhmann. Topology free hidden markov models: Application to background modeling. In *Proceedings of the 8th International Conference on Computer Vision*, volume 1, pages 294–301, 2001.

[121] A. Stolcke. *Bayesian Learning of Probabilistic Language Models*. Ph. D. dissertation, University of California, 1994.

[122] J.F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11–12:625–653, 1999. Special issue on Interior Point Methods (CD supplement with software).

[123] J. Takami and S. Sagayama. A successive state-splitting algorithm for efficient allophone modeling. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, volume 1, pages 573–576, 1992.

[124] B. Taskar, C. Guestrin, and D. Koller. Max-margin markov networks. In *Advances in Neural Information Processing Systems (NIPS 2003)*, Vancouver, Canada, 2004. Winner of the Best Student Paper Award.

[125] F. Thollard, P. Dupont, and C. de la Higuera. Probabilistic DFA Inference using Kullback-Leibler Divergence and Minimality. In *Seventeenth International Conference on Machine Learning*, pages 975–982. Morgan Kauffman, 2000.

[126] H. Trebbe. Errata in rabiner's hmm-tutorial. Technical report, University of Münster, May 1995.

[127] Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, page 104, New York, NY, USA, 2004. ACM Press.

[128] S. van Dongen. A cluster algorithm for graphs. Technical Report INS-R0010, Centrum voor Wiskunde and Informatica, Amsterdam, Netherlands, May 2000.

[129] R. Vanderbei. LOQO: An interior point code for quadratic programming. Technical Report SOR 94-15, Princeton University, 1994.

[130] Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, New York, 1995.

[131] Vladimir Vapnik and Olivier Chapelle. Bounds on error expectation for support vector machines. *Neural Computation*, 12(9):2013–2036, 2000.

[132] Fabien Viger and Matthieu Latapy. Efficient and simple generation of random simple connected graphs with prescribed degree sequence. In *COCOON*, pages 440–449, 2005.

[133] D.M. Warme, P. Winter, and M. Zachariasen. *Advances in Steiner Tree*, chapter Exact Algorithms for Plane Steiner Tree Problems: A Computational Study, pages 81–116. Kluwer Academic Publishers, 2000.

[134] Scott White and Padhraic Smyth. Algorithms for estimating relative importance in networks. *Proceedings of the ninth ACM SIGKDD International Conference on Knowledge Discovery and Data mining*, pages 266–275, 2003.

[135] Ian H. Witten and Timothy C. Bell. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37(4):1085–1094, 1991.

[136] Linli Xu, Dana Wilkinson, Finnegan Southey, and Dale Schuurmans. Discriminative unsupervised learning of structured predictors. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 1057–1064, New York, NY, USA, 2006. ACM Press.

[137] Oksana Yakhnenko, Adrian Silvescu, and Vasant Honavar. Discriminatively trained markov model for sequence classification. In *ICDM '05: Proceedings of the Fifth IEEE International Conference on Data Mining*, pages 498–505, Washington, DC, USA, 2005. IEEE Computer Society.

[138] Luh Yen, Denis Vanvyve, Fabien Wouters, Francois Fouss, Michel Verleysen, and Marco Saerens. Clustering using a random walk-based distance measure. In *Proceedings of the 13th European Symposium on Artificial Neural Networks (ESANN2005)*, pages 317–324, 2005.

[139] Matthew Young-Lai and Frank Wm. Tompa. Stochastic grammatical inference of text database structure. *Machine Learning*, 40(2):111–137, 2000.

[140] H. Zhou. Distance, dissimilarity index, and network community structure. *Physical Review E*, 67(061901), 2003.

[141] H. Zhou. Network landscape from a brownian particles perspective. *Physical Review E*, 67(041908), 2003.

[142] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978.