"Randomized shortest paths and their applications"

García Díez, Silvia

Abstract

In graph analysis, the Shortest Path problem identifies the optimal, most cost effective, path between two nodes. This problem has been the object of many studies and extensions in heterogeneous domains such as: speech recognition, social network analysis, biological sequence alignment, path planning, or zero-sum games among others. Although the shortest path focuses on the optimal cost of reaching a destination node, it does not take into account other useful information contained on the graph, such as the degree of connectivity of two nodes. On the other hand, measures taking connectivity information into account have their own drawbacks, specially when graphs become large. A new family of distances which interpolates between both extremes is introduced by the Randomized Shortest Path (RSP) framework. By spreading randomization through a graph, the RSP leads to applications where some degree of randomness would be desired. Through this work, we try to investigate whether the RSP fr...

Randomized Shortest Paths on graphs and their applications

by Silvia García Díez

Thesis submitted in fulfillment of the requirements for the degree of

Doctor in Engineering Sciences

of the Université catholique de Louvain

Advisor:

Prof. Marco Saerens, Université catholique de Louvain

PhD Jury:

President Prof. Charles Pecheur, Université catholique de Louvain

Prof. Michel Verleysen, Université catholique de Louvain

Prof. Esteban Zimanyi, Université libre de Bruxelles

Prof. François Fouss, Université catholique de Louvain

Prof. Ludovic Denoyer, Université Pierre et Marie Curie, Paris

Dr. Amin Mantrach, Yahoo Labs Barcelona, Barcelona

To my family.

Acknowledgments

First of all, I would like to thank my advisor Professor M. Saerens, who introduced me to the domain of machine learning, and who endured all my questions through these years. I would like to thank him for guiding me on all the topics mentioned in this thesis, for all his wise advice, and for encouraging me to finish this work. I would also like to thank him for all music-related discussions which made me discover more of the French and Belgian music.

I would also like to thank my thesis jury members, Professors M. Verleysen and E. Zimanyi, who have enriched this work with their ideas and suggestions, but also Professors L. Denoyer and F. Fouss for accepting being part of this jury and reviewing my work. I cannot forget to mention Dr. A. Mantrach who helped me run my initial experiments in this domain. Finally, I would like to thank Prof. C. Pecheur for accepting being the president of this jury.

I cannot forget my colleagues from the UCL, with whom I spent many great moments: from the lecturers from my doctoral school and their many useful explanations and insights, my coauthors, to my wonderful colleagues both from the Management School and Engineering School of the UCL. I will keep very precious memories of our times together.

And last but not least, I would like to thank my family and friends for supporting me in this adventure, and for keeping the contact despite the distance, and encouraging me to accomplish this thesis. A special mention must be made to my beloved husband who always supported me in times of frustration, and joy, and who was always by my side.

Thanks for everything you shared with me, and everything I learnt from you.

This thesis was accomplished thanks to the support of the FSR funds from the UCL and a FRIA grant from the Belgian Ministry (FNRS).

Contents

List of Figures

List of Tables

Preface

This thesis is the result of a close collaboration with other colleagues and Universities. Below there is a list of the published articles, together with the names of the co-authors:

1. The work presented in Chapter 2 has been published in:

   Silvia García-Díez, François Fouss, Masashi Shimbo, and Marco Saerens. Normalized sum-over-paths edit distances. In *2010 20th International Conference on Pattern Recognition (ICPR)*, pages 1044–1047. IEEE, 2010.

   Silvia García-Díez, François Fouss, Masashi Shimbo, and Marco Saerens. A sum-over-paths extension of edit distances accounting for all sequence alignments. *Pattern Recognition*, 44(6):1172–1182, 2011a.

   - Silvia García-Díez is with the Unité de Systèmes d'Information (ISYS) from the Université catholique de Louvain, Louvain-la-Neuve, Belgium.
   - François Fouss is with the Louvain School of Management from the Université catholique de Louvain, Mons, Belgium.
   - Masashi Shimbo is with the Graduate School of Information Science from the Nara Institute of Science and Technology, Nara, Japan.
   - Marco Saerens is with the Unité de Systèmes d'Information (ISYS) from the Université catholique de Louvain, Louvain-la-Neuve, Belgium.

2. The work presented in Chapter 3 has been published in:

   Silvia García-Díez, Eric Vandenbussche, and Marco Saerens. A continuous-state version of discrete randomized shortest-paths, with application to path planning. In *50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC)*, pages 6570–6577. IEEE, 2011c.

   - Eric Vandenbussche is with the Université libre de Bruxelles, Bruxelles, Belgium.

3. The work presented in Chapter 4 has been published in:

   Silvia García-Díez, Jérôme Laforge, and Marco Saerens. Rminimax: An optimally randomized minimax algorithm. *IEEE Transactions on Cybernetics*, 43(1):385–393, 2013.

   - Jérôme Laforge is with the Université catholique de Louvain, Louvain-la-Neuve, Belgium.

4. The work presented in Chapter 5 has been published in:

   Silvia García-Díez, Mathieu Senelle, François Fouss, and Marco Saerens. A simple-cycles weighted kernel based on harmony structure for similarity retrieval. In *12th International Society for Music Information Retrieval Conference (ISMIR)*, pages 61–66, 2011b.

   - Mathieu Senelle is with the Louvain School of Management from the Université catholique de Louvain, Mons.

We would like to specially thank (in memoriam) Eric Vandenbussche whose valuable help with the mathematical developments of Chapter 3 was essential for this work.

List of Contributions

The original contributions presented in this thesis are summarized as follows:

- Chapter 1 introduces an alternative, more intuitive, derivation of the forward and backward variables in the context of the Sum-over-Paths framework, and its relation to the main quantities of interest.
- Chapter 2 introduces four novel methods based on the Sum-over-Paths framework for Approximate String Matching: (i) the Sum-over-Paths edit distance, (ii) the Sum-over-Paths common subsequences, (iii) the Normalized Sum-over-Paths edit distance, and (iv) the Normalized Sum-over-Paths common subsequences. These measures provide a model-independent technique for computing similarity by taking all alignments into account. They also avoid noisy measures by favoring relevant subsequences of nearly-optimal alignments. Furthermore, their normalized versions overcome any normalization issue. These measures have been proven to outperform other ASM techniques through empirical validation.
- Chapter 3 provides an optimal randomized policy based on the Sum-over-Paths framework for solving continuous-state path planning problems with multiple sources and multiple goals. It introduces a diffusion parameter for controlling the trade-off between exploration and exploitation, and it shows some interesting links between biased random walks on a graph (discrete RSP) and continuous-state Feynman-Kac diffusion processes.
- Chapter 4 provides a novel global optimal strategy based on the Sum-over-Paths framework given a level of entropy for simulating the AI in two-player zero-sum games. Although the notion of entropy has been widely used for controlling randomness in AI, this new method spreads the entropy over full strategies, instead of single moves.
- Chapter 5 provides a similarity measure for songs based on the repetitive harmonic features of songs. This similarity measure deals with large

structural changes in chord progression which are extracted as cycles from a graph and is computed by means of a kernel function. This approach for harmonic comparison is new in the domain. It also exploits a novel source of user-generated data that is readily available on the Internet.

All the code and data sets used in this work are available at `http://github.com/silviagdiez/thesis`.

Notations

$\mathbf{x}$	A vector
G	A graph
$\mathbf{s}$	A string $\mathbf{s} = s_1 s_2 ... s_n$ of length n is a concatenation of n symbols, s_i.
$c_{kk'}$	Local cost of going from state k to state k'.
$s_{kk'}$	Local similarity between states k and k'.
c_{ins}	Local cost of inserting a symbol s_i into string $\mathbf{s}$.
c_{del}	Local cost of deleting a symbol s_i from string $\mathbf{s}$.
c_{repl}	Local cost of replacing symbol s_i by symbol s_j from string $\mathbf{s}$.
$\wp$	A path on a graph.
$C(\wp)$	Total cost of a path $\wp$, i.e., the sum of all individual costs $c_{kk'} \in \wp$.
$S(\wp)$	Total simlarity of a path $\wp$, i.e., the sum of all individual similarities $s_{kk'} \in \wp$.
$P(\wp)$	Probability of a path $\wp$.
$\lvert\mathbf{s}\rvert$	Size of string $\mathbf{s}$.
$\mathrm{Pred}(k)$	Predecessors of state k.
$\mathrm{Succ}(k)$	Successors of state k.
$\mathcal{Z}$	Partition function.
$\mathcal{P}$	Set of all possible paths between node 1 and node n.
H_0	Entropy of paths on a graph.
θ	Inverse temperature related to H_0.
$\mathbb{E}[X]$	Expectation of the random variable X.
d_{SoP}	Sum-over-Paths edit distance.
s_{SoP}	Sum-over-Paths common subsequence.
z_{1k}	Forward variable for state k.
z_{kn}	Backward variable for state k.

$z^f_{x,y}$	Forward variable for state $k = (x, y)$ in the continuous RSP.
$z^b_{x,y}$	Backward variable for state $k = (x, y)$ in the continuous RSP.
z^f_k	Forward variable for state k.
z^b_k	Backward variable for state k.
$d(\mathbf{s}_1, \mathbf{s}_2)$	Distance between strings $\mathbf{s}_1$ and $\mathbf{s}_2$.
$d_n(\mathbf{s}_1, \mathbf{s}_2)$	Normalized distance between strings $\mathbf{s}_1$ and $\mathbf{s}_2$.
$\mathbf{K}$	Kernel matrix
$V(x, y)$	Cost density at (x, y)
D	Diffusion constant
s	Total displacement along a trajectory
ϵ	Net displacement along a trajectory
$\rho_t(\mathbf{r})$	Particle density at time t and position $\mathbf{r} = (x, y)$
$\mathbf{j}(\mathbf{r}, t)$	Particle flow
$\partial\Omega$	Region boundary
γ	Mobility coefficient
$\mathbf{f}$	External force
$\mathbb{E}_W$	Expectation according to the Wiener measure
$\delta(x - x_f)$	Delta of Dirac at position $x = x_f$
$\delta(x, x_f)$	Delta of Kronecker at position $x = x_f$
$\overline{n}(x, y)$	Expected number of visits at position (x, y)
$\overline{C}$	Expected cost
$\rho^*_t(x, y)$	Optimal probability density of finding the random walker in position (x, y) at time t when starting from some position $\rho^*_0(x, y) = \delta(x - x_0)\delta(y - y_0)$
$\mathbf{G}$	Gram matrix
π_i	Player i
$\mathcal{N}$	Set of final statuses for a game
V	Set of vertices of a graph
E	Set of edges of a graph
C	Simple cycle
v_i	Vertex i of a graph
e_i	Edge i of a graph
λ_i	Label i of a transition in a simple cycle

Chapter 1

Introduction

Contents

In graph analysis, the *Shortest Path problem* identifies the optimal, most cost effective, path between two nodes as shown in Figure 1.1. This problem has been the object of many studies and extensions in heterogeneous domains such as: speech recognition (Jelinek, 1997), social network analysis (Wasserman and Faust, 1994), biological sequence alignment (Durbin et al., 1998), path planning (LaValle, 2006), or zero-sum games (Adelson-Velsky et al., 1988) among others. Although the shortest path focuses on the optimal cost of reaching a destination node, it does not take into account other useful information contained on the graph, such as the degree of connectivity of two nodes.

New measures which also include the connectivity information were proposed: (i) the *resistance distance* (Klein and Randic, 1993) based on electri-

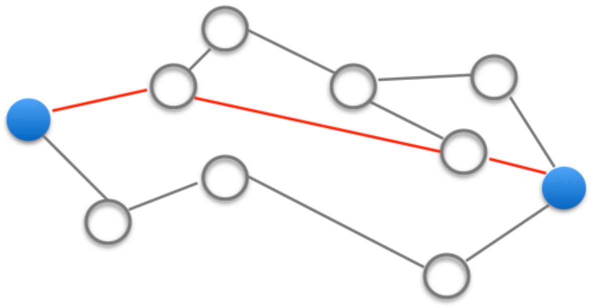


Figure 1.1: Shortest path with a cost of 3 units (marked in red) between an initial and a destination node (marked in blue).

cal networks theory measures distance between nodes based on indirect links, considering highly linked nodes as more similar; and (ii) the *commute-time distance* (Gobel and Jagers, 1974) which is defined as the expected length of paths that a random walker can follow between a pair of nodes and exploits, therefore, all paths linking both nodes. However, the resistance distance and the commute-time distance are not valid measures when the graph becomes excessively large and contains a highly number of connections of all lengths (see, e.g., Luxburg et al. (2010)).

How do we then overcome the limitations of the shortest path – which does not exploit the full information on a graph–, and the resistance or commute-time distance – which are not valid in large graphs–? A new family of distances which "interpolates" between both extremes (i.e., either taking only the optimal path, or just looking at the connectivity) is introduced by the *Randomized Shortest Path framework* (RSP) from Saerens et al. (2009).

The RSP framework exploits sub-optimal paths that are not too far from the shortest path, while taking into account the connectivity between two nodes through multiple paths. By spreading randomization through a graph, the RSP leads to applications where some degree of randomness could be desired, e.g.:

- routing problems with changing environment where some degree of exploration is needed to identify better paths;

- routing problems where some load balancing is needed (in Figure 1.2, e.g., two paths (in red) are used to pass messages between the blue nodes);
- game strategies where randomization is an asset to avoid an opponent (in Fig 1.3, e.g., an opponent (in red) must be avoided by following alternative paths (in blue));
- in dissimilarity measures where the use of multiple paths increases the robustness of the measure.

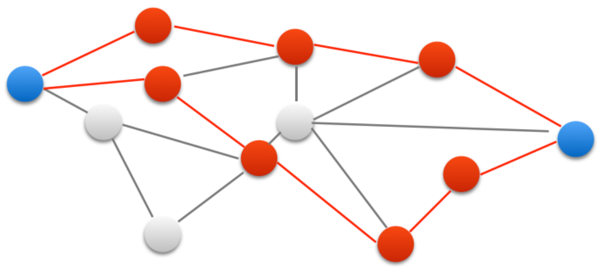


Figure 1.2: Shortest path with a cost of 3 units (marked in red) between an initial and a destination node (marked in blue).

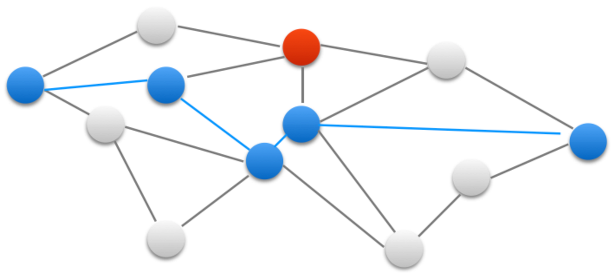


Figure 1.3: Shortest path with a cost of 3 units (marked in red) between an initial and a destination node (marked in blue).

As discussed in (Saerens et al., 2009), the idea of randomizing the policy was introduced by (Achbany et al., 2006, 2008) in the context of reinforcement learning and was inspired by the entropy rate of an ergodic Markov chain defined in information theory (see, e.g., Cover and Thomas (1991)). This previous work was focused on optimal routing of an agent through a network, where some degree of exploration of the network is desired (controlled by the entropy). For this, the sum of the local entropies on all nodes was fixed and the optimal policy for this fixed level of entropy was computed through a value-iteration-like algorithm.

In (Saerens et al., 2009), instead of fixing the local entropy defined at each node as in (Achbany et al., 2006, 2008), the *global entropy spread over the whole network* is fixed. While this difference seems a priori insignificant, it appears that constraining the global spread entropy of the network is more natural. Clearly, the nodes that need a large spread are difficult to determine in advance, and the model has to distribute the entropy by optimizing it globally all over the network. It was shown in (Saerens et al., 2009) that the optimal randomized policies can be found by solving a simple system of linear equations. In the same paper, the authors showed that when the graph is acyclic, the expected cost as well as the policy can be computed efficiently from two simple recurrence relations. This fact is exploited in Chapters 2, 3, and 4, where applications needing a certain degree of randomness, yet optimal, are presented.

Chapter 2 could be of application in Optical Character Recognition (OCR), where specific device encodes the movements of a user's writing, and identifies the particular letters or symbols on the paper. This is useful for transforming analog input into digital format for people who takes notes on paper in their daily lives, but wish to keep an electronic record of it. Another possible application for the technique presented in Chapter 2, is comparison of biological sequences of proteins. Sequence alignment in bioinformatics provides a way to identify similar regions in two proteins, which can be a sign of functional, structural, or evolutionary relation between them. This is useful for a better understanding of these sequences and creation of targeted treatments for certain illnesses.

Chapter 3 presents an algorithm of path planning that allows an agent to move following random smooth trajectories. Path planning is a well-known problem in robotics, where an agent needs to travel from one point to another, sometimes exploring its environment, and avoiding obstacles in a safe way. Although an initial idea of the environment is needed, by following random paths, the agent can improve its knowledge of the environment, and readjust if necessary.

Artificial intelligence is commonly used in games to mimic a rival that can perform with a certain strength. Finding a method that simulate the human behavior of a medium-strength player can be a tricky task. Chapter 4 presents such a method, by allowing random strategies adapted to a certain strength.

The **research question** that we try to investigate through this work is whether the RSP framework can be applied to different domains where randomization is useful, and either solve an existing problem with a new approach, or prove to outperform existing methods.

More specifically, the research questions of each chapter will be:

- Chapter 2: investigate whether a robust distance measure can be obtained by extending the RSP framework, and its relative performance regarding other state of the art distances.
- Chapter 3: investigate whether the continuous counterpart of the RSP framework can provide smooth, sub-optimal trajectories in path planning.
- Chapter 4: investigate whether the RSP framework can be applied to zero-sum two-player games, to simulate an Artificial Intelligent player which has a more human-like behavior by spreading entropy through the whole game tree, and its comparison with similar algorithms.

Additionally, Chapter 5, which is not an application of the RSP framework, investigates how to build a robust Music Information Retrieval system based on chord progressions and its relative performance regarding other distances. The purpose here is to build a more intelligent music recommendation system that focuses on the mood of songs, and how the tension is built through harmonic sequences, regardless of genre or other parameters.

Data in form of graphs

The reader should note that the applications presented in this work take as input data which is naturally expressed in form of graphs. For instance, the lattice that contains all editing paths from Chapter 2 is commonly used for string similarity; path planning in 2D in Chapter 3 uses a representation of space as a grid composed by connected nodes through which an agent can move; the game tree from Chapter 4 is a well-known structure in game theory, where an initial state of the game branches into different possibilities as the players make their moves. Chapter 5, however, makes the assumption that we can convert a sequential chord progression into a graph with transitions between chords, as a means of extracting simple cycles, which are the basis of a similarity measure. This hypothesis is validated by the fact that the similarity measure performs well, and that this type of data is well suited for the simple cycle extraction.

Main contributions

In this chapter we present an alternative, more intuitive, derivation of the forward and backward variables in the context of the Sum-over-Paths framework, and its relation to the main quantities of interest.

1.1 The discrete Randomized Shortest Path framework

The Randomized Shortest Path (RSP) framework addresses the problem of designing transitions probabilities (or policy) on a Markov chain so to minimize the expected cost of reaching a destination, while keeping some level of randomness (or entropy) spread in the graph. This framework exploits the idea of path as a frequent motif on a graph, which is used as a basis for extracting some quantities of interest, such as the expected mean number of passages on a node, or a fundamental matrix. We show in future chapters, how to apply these quantities based on paths to several domains, in order to compute more robust dissimilarity measures, more human-like AI algorithms, or to generate sub-optimal random movements on motion planning.

1.1.1 A short reminder about Markov chains

This Section defines a random walk on a graph, and its link with the RSP framework. Let us first recall some basics on Markov chains from Norris (1997): a discrete Markov chain is a system composed by states and transitions, as shown in Figure 1.4. Each transition between two states i and j has an associated probability p_{ij} (the transition probability) according to the Markov stochastic process. The probability of being on a certain state depends only on the previous state, but not on the earlier states, making this stochastic process memoryless. We define an absorbing Markov chain, as a model in which exists at least one absorbing node, i.e., $\{\exists v_i, \forall j \neq i \in V | p_{ij} = 0\}$.

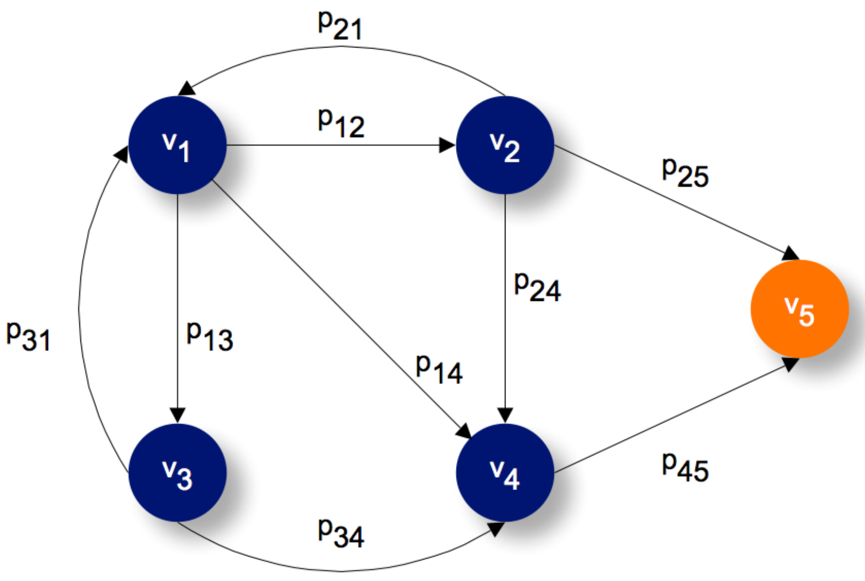


Figure 1.4: Example of a first-order discrete Markov chain with one absorbing node (in orange).

Doyle and Snell (1984) denote the matrix containing all transition probabilities of an absorbing Markov chain with n states (of which r are transient, i.e., non-absorbing) as:

$$\mathbf{P} = \left(\begin{array}{c|c} \mathbf{Q} & \mathbf{R} \\ \hline \mathbf{0} & \mathbf{I} \end{array}\right) \tag{1.1}$$

where the states have been re-ordered and:

- $\mathbf{Q}$ is a $r \times r$ matrix containing the transition probabilities for transient states.
- $\mathbf{R}$ is a $r \times (n-r)$ matrix containing the transition probabilities to absorbing states.
- $\mathbf{I}$ is the $(n-r) \times (n-r)$ identity matrix.
- $\mathbf{0}$ is a $(n-r) \times r$ matrix containing all zeros, as there is no outgoing transition probability from an absorbing state.

The above notation is called the *canonical form* of an absorbing Markov chain. If we define the probability transition matrix for the example of the Markov chain from Figure 1.4 we would obtain:

$$\mathrm{P} = \left(\begin{array}{cccc|c} 0 & p_{12} & p_{13} & p_{14} & 0 \\ p_{21} & 0 & 0 & p_{24} & p_{25} \\ p_{31} & 0 & 0 & p_{34} & 0 \\ 0 & 0 & 0 & 0 & p_{45} \\ \hline 0 & 0 & 0 & 0 & 1 \end{array}\right) \tag{1.2}$$

We define the *fundamental matrix* of an absorbing Markov chain $\mathbf{P}$ as the matrix

$$\mathbf{N} = (\mathbf{I} - \mathbf{Q})^{-1} = \mathbf{I} + \mathbf{Q} + \mathbf{Q}^2 + \cdots \tag{1.3}$$

where each element n_{ij} corresponds to the expected number of times the chain is in state s_j after starting in state s_i.

We can use this fundamental matrix to compute the main quantities of interest, such as the *time to absorption* which can be computed as $\mathbf{t} = \mathbf{Nc}$, where $\mathbf{c}$ is a vector of all ones, and the t_i element represents the expected number of steps before absorption when starting in state s_i.

On the other hand, we say that a Markov chain is ergodic if we can reach any state from any other state in a finite number of steps. The example of Figure 1.4 is not ergodic, as it is not possible to reach any state from v_5. The fundamental matrix of ergodic Markov chains is computed differently, but the same principle applies, and we can easily obtain from it quantities such as the mean passage rate or the average first-passage time.

The analogy between the fundamental matrix in Markov chains and the partition function of the RSP, which is a component of that matrix, is studied in Section 1.4.

1.2 Related work

Apart from the work from Achbany et al. (2006, 2008); Saerens et al. (2009), and some others in game theory (see for instance (Osborne, 2004)) or Markov games (Littman, 1994), very few optimal randomized strategies have been exploited in the context of shortest-path problems. There is, however, one exception: the model from Akamatsu (1996), who designed a randomized policy for routing traffic in transportation networks. In transportation science, randomized strategies are called *stochastic traffic assignments* and, within this context, Akamatsu's model is the model of reference. This framework, as well as (Saerens et al., 2009), are inspired by Akamatsu's model.

Let us also mention some papers that are related to path randomization, and therefore to the present work. The entropy of the paths (or trajectories) connecting an initial and an absorbing destination node of an absorbing Markov chain was studied in Ekroot and Cover (1993). In this paper, the authors provide formulas allowing to compute the entropy needed to reach the destination node. In (Tahbaz and Jadbabaie, 2006) a one-parameter family of algorithms that recover both the procedure for finding shortest paths as well as the iterative algorithm for computing the average first-passage time in a Markov chain is introduced. However, having a heuristic foundation, it is not based on the optimization of a well-defined cost function.

In another context, Todorov (Todorov, 2006) studied a family of Markov decision problems that are linearly solvable, that is, for which a solution can be computed by solving a matrix eigenvector problem. In order to make this possible, Todorov assumes a special form of control for the transition probabilities, which recasts the problem of finding the policy into an eigenvector problem. In (Boyd et al., 2004) a Markov chain that has the fastest mixing properties is designed, while (Sun et al., 2006) discuss its continuous-time counterpart.

In a completely different framework, uninformed random walks, based on maximizing the long-term entropy (Delvenne and Libert, 2011; Tomlin, 2003), have recently been proposed as an alternative to the standard PageRank algorithm. Finally, notice that some authors tackled the problem of designing ergodic (non-absorbing) Markov or semi-Markov chains (see, e.g., Sahner et al. (2012)) in a maximum-entropy framework (see for instance (Girardin, 2004; Girardin and Limnios, 2004) and the references therein).

1.3 Background and notations

This section provides a short account of the discrete randomized shortest path (RSP) framework (Saerens et al., 2009; Yen et al., 2008), in the context of a single-source single-destination problem, which was initially inspired by a stochastic traffic assignment model (Akamatsu, 1996). It will be shown that this RSP framework allows solving minimal-cost problems in a graph by means of simple linear algebra operations.

Let G be a directed graph containing a source node with index 1 and a destination or goal node with index n (we assume $n \neq 1$). Moreover, the goal node is absorbing: once node n is reached, the path stops, i.e., there is no outgoing arc from n. A non-negative local cost $c_{kk'} \geq 0$ is associated to each of the arcs. We consider graphs with no self-loops, which implies that the cost of remaining in the same state is infinite, $c_{kk} = \infty$ for all k. Similarly, an infinite cost is assumed when there is no arc between node k and k'. If there are many destination nodes, the following trick can be used: a dummy node n is created and a zero-cost arc between each destination node and the dummy node n is added. Furthermore, the transition costs from the destination node are all $c_{nk} = \infty \; \forall k$, i.e., the random walker dissapears when reaching node n.

We now adopt a **sum-over-paths formalism** (see Mantrach et al. (2010)):

let us further denote by $\mathcal{P}$ the set of all paths (including cycles) that go from 1 to n. Each path $\wp \in \mathcal{P}$ is composed by a sequence of arcs $k \to k'$ that ties the source to the destination node. Moreover, let the total cost $C(\wp)$ of a path $\wp$ be the sum of these local costs along $\wp$. The path randomization will be driven by global performance: a probability is assigned to each path, favoring nearly-optimal paths having a low cost $C(\wp)$. Therefore, optimal or slightly nearly-optimal paths are assigned a high probability, while paths leading to a high cost are penalized.

The *Shannon entropy* defined as

$$H_0 = -\sum_{\wp \in \mathcal{P}} \mathrm{P}(\wp) \ln \mathrm{P}(\wp) \tag{1.4}$$

is used for controlling this penalization of expensive paths via the assigned probability distribution. The parameter H_0 controls the degree of randomness, or exploration, in the graph and is related to the *inverse temperature* of the graph, $\theta = 1/T > 0$.

Let the initial, reference, policy be uniform (pure exploration – the costs are not taken into account), $p_{kk'}^{(0)} = 1/n_k$ (except for $k = n$ for which $p_{kk'}^{(0)} = 0$), where n_k is the out-degree of node k. Let us now seek the optimal probability distribution $\mathrm{P}^*(\wp)$ on the set of paths $\mathcal{P}$ – assumed to be independent – minimizing the expected cost for reaching the destination node from the source node (exploitation) while maintaining a constant relative entropy H_0 with respect to this reference policy (exploration). The problem to solve is

$$\left| \begin{array}{ll} \underset{\mathrm{P}(\wp)}{\text{minimize}} & \sum_{\wp \in \mathcal{P}} \mathrm{P}(\wp) C(\wp) \\ \text{subject to} & \sum_{\wp \in \mathcal{P}} \mathrm{P}(\wp) \ln(\mathrm{P}(\wp)/\mathrm{P}^{(0)}(\wp)) = H_0 \end{array} \right. \tag{1.5}$$

where $\mathrm{P}^{(0)}(\wp)$ is the probability of following path $\wp$ when using the reference policy, i.e., transition probabilities $p_{kk'}^{(0)}$. The sum in (1.5) is defined on all paths $\wp \in \mathcal{P}$. It can be easily shown (Saerens et al., 2009; Mantrach et al., 2010) that the optimal path probability distribution is a **Boltzmann distribution** on the set of paths $\mathcal{P}$,

$$\mathrm{P}^*(\wp) = \frac{\mathrm{P}^{(0)}(\wp)\exp\left[-\theta C(\wp)\right]}{\sum\limits_{\wp' \in \mathcal{P}} \mathrm{P}^{(0)}(\wp')\exp\left[-\theta C(\wp')\right]} \tag{1.6}$$

where $\theta > 0$ is the **inverse temperature** and is directly related to the relative entropy H_0.

This equation provides the **optimal randomized policy** expressed in terms of path probabilities. As expected, shorter paths $\wp$, having small $C(\wp)$, are favored. Indeed, when θ is large, the probability distribution defined by Equation (1.6) is biased towards short (low cost) paths, i.e., when $\theta \to \infty$, the probability distribution peaks around the shortest paths. Let us further define the **partition function** as

$$\mathcal{Z} = \sum_{\wp \in \mathcal{P}} \mathrm{P}^{(0)}(\wp)\exp\left[-\theta C(\wp)\right] \tag{1.7}$$

which is the denominator from Equation (1.6). Clearly, the **expected number of passages** $n_{kk'}$ through arc $k \to k'$ is given by (see Jaynes (1957) or Section 1.6 for an alternative derivation in the sum-over-paths context)

$$n_{kk'} = -\frac{1}{\theta}\frac{\partial(\ln \mathcal{Z})}{\partial c_{kk'}} \tag{1.8}$$

and the expected number of passages per node $n_{k'}$ are given by

$$n_{k'} = \sum_{k \in \mathrm{Pred}(k')} n_{kk'} \tag{1.9}$$

where $\mathrm{Pred}(k')$ is the set of predecessors of node $k' \neq 1$ (the initial node).

For the single-source single-destination problem with independent paths, the optimal policy expressed in terms of path probabilities in Equation (1.6), providing the minimum-cost policy for a constant H_0, can be re-expressed in terms of **local**, **optimal**, **transition probabilities**, i.e., a local first-order

Markov policy (see Saerens et al. (2009))

$$p^*_{kk'} = \frac{n_{kk'}}{\sum\limits_{l \in \text{Succ}(k)} n_{kl}}, \text{ for } k \neq n \tag{1.10}$$

which provides the transition probabilities from every node of interest of the graph. Here, Succ(k) is the set of successors of node k. Other useful measures, such as the expected cost, can also be calculated through the partition function as well-known in statistical physics and as shown, e.g., by Jaynes (1957)

$$\overline{C} = -\frac{1}{\theta}\frac{\partial(\ln \mathcal{Z})}{\partial\theta} \tag{1.11}$$

Let us now show how this partition function, $\mathcal{Z}$, can be computed from the cost matrix and the reference transition probabilities.

1.4 Computation of the partition function

As in Markov chains, there exists a “fundamental matrix” from which all main quantities of interest can be easily computed. In the RSP framework, this fundamental matrix is related to the partition function $\mathcal{Z}$, which as shown by Saerens et al. (2009) and Mantrach et al. (2010), can be computed from the immediate cost matrix, $\mathbf{C}$, and the reference transition matrix, $\mathbf{P}^{(0)}$, containing the $p^{(0)}_{kk'}$. We first define $\mathbf{W}$ as

$$\mathbf{W} = \mathbf{P}^{(0)} \circ \exp[-\theta\mathbf{C}] = \exp[-\theta\mathbf{C} + \ln \mathbf{P}^{(0)}] \tag{1.12}$$

where the logarithm/exponential functions are taken element-wise ($\circ$ is the Hadamard matrix product). Further developments (Saerens et al., 2009) show that the partition function (Equation (1.7)) can be computed as

$$\mathcal{Z} = [(\mathbf{I} - \mathbf{W})^{-1}]_{1n} = [\mathbf{Z}]_{1n} = z_{1n} \tag{1.13}$$

and by analogy with Markov chains, $\mathbf{Z} = (\mathbf{I} - \mathbf{W})^{-1} = \mathbf{I} + \mathbf{W} + \mathbf{W}^2 + \cdots$, will be called the **fundamental matrix** of the RSP. Its elements (k, l) are denoted as z_{kl}. The matrix $\mathbf{Z}$ cumulates the contributions of paths of length 0, length 1, etc. A zero-length path is thus allowed, with zero cost by convention. Furthermore, $\mathbf{Z}$ is always well-defined since $\mathbf{W}$ is sub-stochastic.

1.5 Computation of the main quantities

Let us now show how we can compute the main quantities of interest from the partition function. Our next step aims at computing the **expected number of passages through arc** $k \to k'$. The partial derivative from Equation (1.8) can be readily computed (Saerens et al., 2009; Mantrach et al., 2010) as

$$n_{kk'} = \frac{z_{1k} w_{kk'} z_{k'n}}{z_{1n}} \tag{1.14}$$

where $w_{kk'}$ is element (k, k') of matrix $\mathbf{W}$. From Equation (1.9), **the expected number of visits** to node k' is given by

$$n_{k'} = \frac{z_{1k'} z_{k'n}}{z_{1n}}, \text{ for } k' \neq 1 \tag{1.15}$$

where we used the identity $z_{1k'} = \delta_{1k'} + \sum_{k \in Pred(k')} z_{1k} w_{kk'}$ that can easily be deducted from $\mathbf{Z} = \mathbf{I} + \mathbf{W}\mathbf{Z}$. Moreover, from Equations (1.10) and (1.14), the **optimal transition probabilities** are

$$p^*_{kk'} = \frac{w_{kk'} z_{k'n}}{\displaystyle\sum_{l \in Succ(k)} w_{kl} z_{ln}} = \frac{w_{kk'} z_{k'n}}{z_{kn}} = \frac{z_{k'n}}{z_{kn}} p^{(0)}_{kk'} \exp[-\theta c_{kk'}], \text{ for } k \neq n \tag{1.16}$$

where we used $z_{kn} = \delta_{kn} + \sum_{l \in Succ(k)} w_{kl} z_{ln}$ coming from $\mathbf{Z} = \mathbf{I} + \mathbf{W}\mathbf{Z}$. On the other hand, when $k = n$, $p^*_{kk'} = 0$, since node n is absorbing. Therefore, Equation (1.16) is still valid for $k = n$ as $c_{nk'} = \infty$, thus $w_{nk'} = 0$, and $z_{nn} = 1$.

When $\theta \to \infty$, the $p^*_{kk'}$ encode the minimum-cost policy, while for intermediate values of θ, following Equation (1.5), they define a Markov chain minimizing the expected cost to the destination for a given relative entropy spread in the graph. It can be observed that these optimal transition probabilities (the optimal policy) do not depend on the initial, source, node (node 1) – they only depend on the destination node n. Equation (1.16) therefore defines the **optimal randomized policy** from any source node – it is actually the *local* policy counterpart of Equation (1.6) when paths are considered as independent (Saerens et al., 2009).

Therefore, the one-step ahead probability distribution of finding the random walker in state k' at time step $t+1$ when following the optimal policy (1.16), $\rho^*_{t+1}(k')$, given its distribution $\rho^*_t(k)$ at time t, is

$$\rho^*_{t+1}(k') = \sum_{k=1}^{n} p^*_{kk'} \rho^*_t(k) = \sum_{k=1}^{n} \frac{w_{kk'} z_{k'n}}{z_{kn}} \rho^*_t(k) \tag{1.17}$$

Let us now come back to the computation of the elements of the fundamental matrix, z_{kl}. Since the Equations (1.14-1.17) only involve the first row and the last column of matrix $\mathbf{Z}$, they can be easily computed by solving two systems of linear equations. For the last column, we solve $(\mathbf{I} - \mathbf{W})\mathbf{z}^b = \mathbf{e}_n$, where $\mathbf{z}^b$ is the column vector of so-called *backward variables* ($[\mathbf{z}^b]_k = z_{kn}$). Symmetrically, the column vector of *forward variables*, $\mathbf{z}^f$ (with $[\mathbf{z}^f]_k = z_{1k}$), containing the first row of matrix $\mathbf{Z}$, is provided by $(\mathbf{I} - \mathbf{W})^{\mathrm{T}}\mathbf{z}^f = \mathbf{e}_1$. Thus, $z^f_k = z_{1k}$ and $z^b_k = z_{kn}$. Written element-wise, this reads

$$\begin{cases} z_{11} = 1 + \displaystyle\sum_{k \in Pred(1)} p^{(0)}_{k1} \exp[-\theta c_{k1}]\, z_{1k} \\ z_{1k'} = \displaystyle\sum_{k \in Pred(k')} p^{(0)}_{kk'} \exp[-\theta c_{kk'}]\, z_{1k}, \text{ for } k' \neq 1 \end{cases} \tag{1.18}$$

and for the backward variables,

$$\begin{cases} z_{nn} = 1 + \displaystyle\sum_{k' \in Succ(n)} p^{(0)}_{nk'} \exp[-\theta c_{nk'}]\, z_{k'n} (= 1 \text{ if node } n \text{ is absorbing}) \\ z_{kn} = \displaystyle\sum_{k' \in Succ(k)} p^{(0)}_{kk'} \exp[-\theta c_{kk'}]\, z_{k'n}, \text{ for } k \neq n \end{cases} \tag{1.19}$$

Now, these backward variables z_{kn} given by Equation (1.19) have an interesting, intuitive, interpretation (Mantrach et al., 2010). Consider a new random walk defined by the transition probabilities (with a tilde)

$$\widetilde{p}_{kk'} = p^{(0)}_{kk'} \exp\left[-\theta c_{kk'}\right] \tag{1.20}$$

without any normalization. Since $\theta > 0$, the transition matrix $\widetilde{\mathbf{P}}$, containing the $\widetilde{p}_{kk'}$, is sub-stochastic. This means that, at each time step of the random walk and for each node k, the random walker has a non-zero probability of abandoning the walk equal to $\widetilde{p}_{\text{eva},k} = (1 - \sum_{k' \in S(k)} \widetilde{p}_{kk'})$. We will say that Equation (1.20) defines an **evaporating (or killed) random walk** (ERW) since the probability of seeing the random walker pursuing its quest decreases at each time step. In that case, the backward variable z_{kn} from Equation (1.19) can be interpreted as the *expected number of passages through node n* (Kemeny and Snell, 1976; Taylor and Karlin, 1998) for a random walker starting in node k at $t = 0$ and ending in n. The quantity $\ln z_{kn}$ will act as a potential (see Equation (3.50)).

In the same manner, one can show that the expected cost from Equation (1.11), when starting from the source node and following the optimal policy, is provided by

$$\overline{C} = \sum_{k=1}^{n} \sum_{k' \in Succ(k)} c_{kk'} n_{kk'} = \frac{1}{z_{1n}} \sum_{k=1}^{n} \sum_{k' \in Succ(k)} z_{1k} c_{kk'} w_{kk'} z_{k'n} \tag{1.21}$$

where $\infty. \exp[-\infty] = 0$ by convention and we used Equation (1.14).

1.6 Computation of the important quantities

Let us now present an alternative, more intuitive, derivation of the forward and backward variables in the context of the sum-over-paths formalism (Saerens et al., 2009), and how this relates to the main quantities of interest. For simplification purposes, let us rewrite the probability distribution (Equation (1.6)) on the paths as

$$\mathrm{P}(\wp) = \frac{\exp\left[-\theta\widetilde{C}(\wp)\right]}{\sum\limits_{\wp'\in\mathcal{P}} \exp\left[-\theta\widetilde{C}(\wp')\right]} \tag{1.22}$$

where $\widetilde{C}(\wp) = -\theta C(\wp) + \ln P^{(0)}(\wp)$.

Let us redefine the expected number of passages through node k as (see Equation (1.15))

$$n_k = \sum_{\wp\in\mathcal{P}_{1n}} \delta(\wp;k)\,\frac{\exp[-\theta\widetilde{C}(\wp)]}{\mathcal{Z}} = \sum_{\wp\in\mathcal{P}_{1n}} \delta(\wp;k)\mathrm{P}(\wp) \tag{1.23}$$

where $\delta(\wp;k)$ is an indicator function that counts the number of times that node k is traversed by path $\wp$. Similarly, the expected number of passages through link $k \to k'$ (see Equation (1.14)) can be expressed as

$$n_{kk'} = \sum_{\wp\in\mathcal{P}_{1n}} \delta(\wp;k,k')\,\frac{\exp[-\theta\widetilde{C}(\wp)]}{\mathcal{Z}} = \sum_{\wp\in\mathcal{P}_{1n}} \delta(\wp;k,k')\mathrm{P}(\wp) \tag{1.24}$$

where $\delta(\wp;k,k')$ counts the number of times link $k \to k'$ is traversed by path $\wp$. If the *graph is acyclic*, which we assume for now (e.g., an acyclic lattice), so that a path can only visit a link once, it returns 1 if, respectively, the node k or the link $k \to k'$ is part of path $\wp$, and 0 otherwise. Thus, the total cost

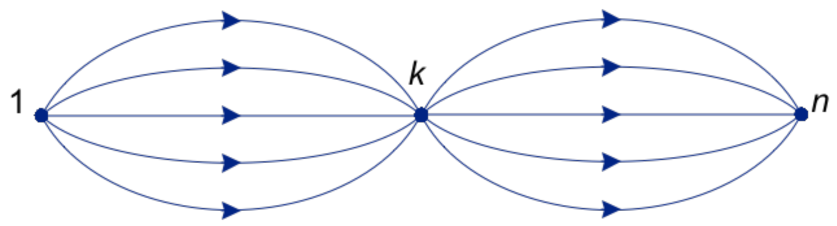


Figure 1.5: The paths $\mathcal{P}_{1n}$ from 1 to n traversing node k can be decomposed into two sub-paths in $\mathcal{P}_{1k}$ and in $\mathcal{P}_{kn}$.

occurred along the path $\wp$ can be computed from the individual costs $c_{kk'}$ by

$$C(\wp) = \sum_{k=1}^{n} \sum_{k'=1}^{n} \delta(\wp; k, k')\, c_{kk'} \tag{1.25}$$

Please refer to Equations (2.6) and (2.10) for the computation of the $\mathbb{E}[C]$.

By generalizing with the same argument, the expectation of any quantity $v_{kk'}$ (instead of the costs $c_{kk'}$) defined on the links is

$$\mathbb{E}[V] = \sum_{\wp \in \mathcal{P}_{1n}} \mathrm{P}(\wp)\, V(\wp) = \sum_{k=1}^{n} \sum_{k'=1}^{n} v_{kk'} n_{kk'} \tag{1.26}$$

where $V(\wp)$ is the sum of the $v_{kk'}$ along the path $\wp$.

Let us now show that, as in hidden Markov models, the average number of passages through each node can easily be expressed in terms of forward and backward variables. Notice first that, since only the paths passing through node k contribute to the sum in Equation (1.23), they can be split into two sub-paths (see Figure 1.5): $\wp_{1k} \in \mathcal{P}_{1k}$ and $\wp_{kn} \in \mathcal{P}_{kn}$. These two sub-paths can be chosen independently since their composition is a valid path, where $\wp_{1k}\wp_{kn} \in \mathcal{P}_{1n}$ is the concatenation of the two paths.

Now, since $\widetilde{C}(\wp) = \widetilde{C}(\wp_{1k}) + \widetilde{C}(\wp_{kn})$ for any $\wp = \wp_{1k}\wp_{kn}$, we easily obtain

$$\begin{aligned}
n_k &= \sum_{\wp \in \mathcal{P}_{1n}} \delta(\wp; k) \, \frac{\exp[-\theta \widetilde{C}(\wp)]}{\mathcal{Z}} && (1.27) \\
&= \frac{1}{\mathcal{Z}} \sum_{\substack{\wp_{1k} \in \mathcal{P}_{1k} \\ \wp_{kn} \in \mathcal{P}_{kn}}} \exp[-\theta \widetilde{C}(\wp_{1k})] \exp[-\theta \widetilde{C}(\wp_{kn})] && (1.28) \\
&= \frac{1}{\mathcal{Z}} \sum_{\wp_{1k} \in \mathcal{P}_{1k}} \sum_{\wp_{kn} \in \mathcal{P}_{kn}} \exp[-\theta \widetilde{C}(\wp_{1k})] \exp[-\theta \widetilde{C}(\wp_{kn})] && (1.29) \\
&= \frac{1}{\mathcal{Z}} \left(\sum_{\wp_{1k} \in \mathcal{P}_{1k}} \exp[-\theta \widetilde{C}(\wp_{1k})] \right) \left(\sum_{\wp_{kn} \in \mathcal{P}_{kn}} \exp[-\theta \widetilde{C}(\wp_{kn})] \right) && (1.30) \\
&= \frac{z_{1k} z_{kn}}{\mathcal{Z}} = \frac{z_{1k} z_{kn}}{z_{1n}} && (1.31)
\end{aligned}$$

where we defined the *forward variable* z_{1k} and the *backward variable* z_{kn} as

$$z_{1k} = \sum_{\wp \in \mathcal{P}_{1k}} \exp[-\theta \widetilde{C}(\wp)] \tag{1.32}$$

$$z_{kn} = \sum_{\wp \in \mathcal{P}_{kn}} \exp[-\theta \widetilde{C}(\wp)] \tag{1.33}$$

and it is clear from Equation (1.31) that $\mathcal{Z} = z_{1n}$, which is equivalent to the result in Equation (2.4).

Interesting enough, Equation (1.31) still holds in the case of *arbitrary weighted directed graphs*, and not only for acyclic lattices. In that situation, paths might contain cycles and the decomposition $\wp = \wp_{1k}\wp_{kn}$ is no more unique: a single path $\wp$ can be decomposed in $\wp_{1k}\wp_{kn}$ in as many ways as the number of times $\wp$ passes through node k. Therefore, each path $\wp \in \mathcal{P}_{1n}$ is counted several times in the sum of Equation (1.28): it appears as many times as the path $\wp$ can be decomposed in $\wp = \wp_{1k}\wp_{kn}$. But this quantity corresponds to the number of times node k appears in path $\wp$ which, in turn, is exactly equal to $\delta(\wp; k)$. Therefore, the passage from Equation (1.27) to Equation (1.28) remains valid for general graphs containing cycles.

The same reasoning holds for the average number of passages through each

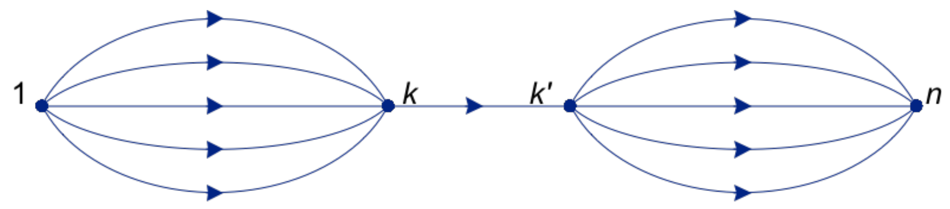


Figure 1.6: The paths $\mathcal{P}_{1n}$ from 1 to n traversing link $k \to k'$ can be decomposed into three sub-paths respectively in $\mathcal{P}_{1k}$, in $\mathcal{P}_{kk'}$, and in $\mathcal{P}_{k'n}$.

link. Indeed, only the paths passing through link $k \to k'$ contribute to the sum in Equation (1.24). They can therefore be split into three sub-paths (see Figure 1.6): $\wp_{1k} \in \mathcal{P}_{1k}$, $\wp_{kk'} \in \mathcal{P}_{kk'}$ and $\wp_{k'n} \in \mathcal{P}_{k'n}$. As before, the three sub-paths can be chosen independently since their composition is a valid path, $\wp_{1k}\wp_{kk'}\wp_{k'n} \in \mathcal{P}_{1n}$.

Since $\widetilde{C}(\wp) = \widetilde{C}(\wp_{1k}) + \widetilde{C}(\wp_{kk'}) + \widetilde{C}(\wp_{k'n})$, we obtain

$$\begin{aligned}
n_{kk'} &= \sum_{\wp \in \mathcal{P}_{1n}} \delta(\wp; k, k') \frac{\exp[-\theta \widetilde{C}(\wp)]}{\mathcal{Z}} \\
&= \frac{1}{\mathcal{Z}} \sum_{\substack{\wp_{1k} \in \mathcal{P}_{1k} \\ \wp_{kk'} \in \mathcal{P}_{kk'} \\ \wp_{k'n} \in \mathcal{P}_{k'n}}} \exp[-\theta \widetilde{C}(\wp_{1k})] \exp[-\theta \widetilde{C}(\wp_{kk'})] \exp[-\theta \widetilde{C}(\wp_{k'n})] \\
&= \frac{1}{\mathcal{Z}} \sum_{\wp_{1k} \in \mathcal{P}_{1k}} \sum_{\wp_{kk'} \in \mathcal{P}_{kk'}} \sum_{\wp_{k'n} \in \mathcal{P}_{k'n}} \exp[-\theta \widetilde{C}(\wp_{1k})] \exp[-\theta \widetilde{C}(\wp_{kk'})] \exp[-\theta \widetilde{C}(\wp_{k'n})] \\
&= \frac{1}{\mathcal{Z}} \left(\sum_{\wp_{1k} \in \mathcal{P}_{1k}} \exp[-\theta \widetilde{C}(\wp_{1k})] \right) \left(\sum_{\wp_{kk'} \in \mathcal{P}_{kk'}} \exp[-\theta \widetilde{C}(\wp_{kk'})] \right) \left(\sum_{\wp_{k'n} \in \mathcal{P}_{k'n}} \exp[-\theta \widetilde{C}(\wp_{k'n})] \right) \\
&= \frac{z_{1k} z_{kk'} z_{k'n}}{\mathcal{Z}} = \frac{z_{1k} \exp[-\theta c_{kk'}] p_{kk'} z_{k'n}}{z_{1n}} \qquad (1.34)
\end{aligned}$$

Once more, by the same argument, it can be shown that the Equation (1.34) still holds for graphs containing cycles.

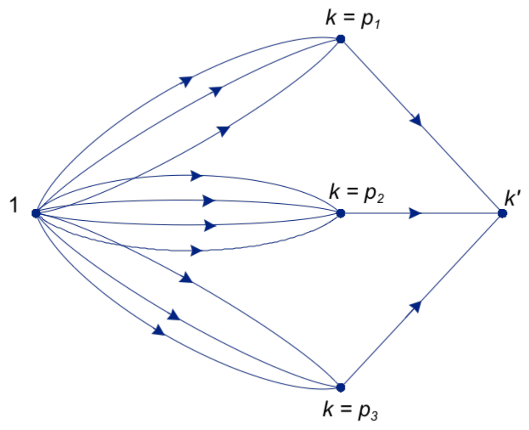


Figure 1.7: Forward recurrence: first compute the paths from node 1 to node k. Then, jump from node k to node k'. The nodes k are the predecessors of node k', $k \in \text{Pred}(k')$.

1.7 Computation of the recurrence relations

Let us now derive the **recurrence relations** for computing these forward/backward variables efficiently. We first investigate the forward variables. For $k' \neq 1$, so that k' has necessarily a predecesor, $k \in \text{Pred}(k')$ (see Figure 1.7), the path $\mathcal{P}_{1k'}$ can be decomposed into $\mathcal{P}_{1k} \in \mathcal{P}_{1k'}$ and $\mathcal{P}_{kk'} \in \mathcal{P}_{1k'}$:

$$\begin{aligned}
z_{1k'} &= \sum_{\wp_{1k'} \in \mathcal{P}_{1k'}} \exp[-\theta \widetilde{C}(\wp_{1k'})] \\
&= \sum_{k \in \text{Pred}(k')} \sum_{\wp_{1k} \in \mathcal{P}_{1k}} \exp[-\theta(\widetilde{C}(\wp_{1k}) + \widetilde{C}(\wp_{kk'}))] \\
&= \sum_{k \in \text{Pred}(k')} \left(\sum_{\wp_{1k} \in \mathcal{P}_{1k}} \exp[-\theta \widetilde{C}(\wp_{1k})] \right) \exp[-\theta \widetilde{C}(\wp_{kk'})] \\
&= \sum_{k \in \text{Pred}(k')} \exp[-\theta \widetilde{C}(\wp_{kk'})]\, z_{1k} \\
&= \sum_{k \in \text{Pred}(k')} \exp[-\theta c_{kk'} + \ln p_{kk'}^{(0)}]\, z_{1k} \qquad (1.35)
\end{aligned}$$

For $k' = 1$, i.e., z_{11}, either $\mathcal{P}_{11}$ is the zero-length path and its contribution is equal to 1, or its length is greater than 0 and it has a set of predecessors.

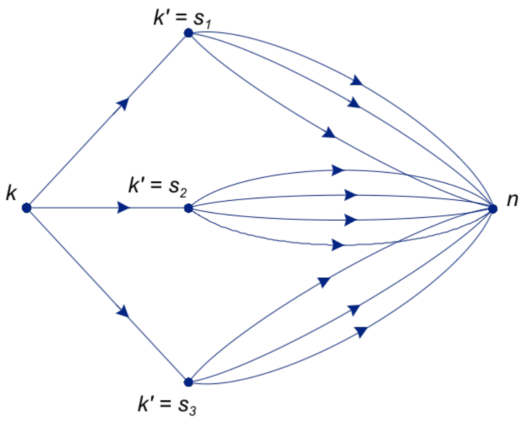


Figure 1.8: Backward recurrence: first jump from node k to node k'. Then, compute the paths from node k' to node n. The nodes k' are the successors of node k, $k' \in \mathrm{Succ}(k)$.

Thus, the same reasoning from Equation (1.35) applies here:

$$
\begin{aligned}
z_{11} &= \sum_{\wp_{11} \in \mathcal{P}_{11}} \exp[-\theta \widetilde{C}(\wp_{11})] \\
&= \sum_{\wp_{11}^0 \text{ of zero length}} \exp[-\theta(\widetilde{C}(\wp_{11}^0)] + \sum_{k \in \mathrm{Pred}(1)} \left(\sum_{\wp_{1k} \in \mathcal{P}_{1k}} \exp[-\theta \widetilde{C}(\wp_{1k})] \right) \\
&= 1 + \sum_{k \in \mathrm{Pred}(1)} \exp[-\theta c_{k1} + \ln p_{k1}^{(0)}] z_{1k} \qquad (1.36)
\end{aligned}
$$

Symmetrically, the same calculation can be performed for the backward variable. The recurrence relations for the forward and backward variables are thus

$$
\begin{cases}
z_{11} &= 1 + \displaystyle\sum_{k \in \mathrm{Pred}(1)} \exp[-\theta c_{k1} + \ln p_{k1}^{(0)}] z_{k1} \\
z_{1k'} &= \displaystyle\sum_{k \in \mathrm{Pred}(k')} p_{kk'}^{(0)} \exp[-\theta c_{kk'}]\, z_{1k}
\end{cases} \qquad (1.37)
$$

$$
\begin{cases}
z_{nn} &= 1 \\
z_{kn} &= \displaystyle\sum_{k' \in \mathrm{Succ}(k)} p_{kk'}^{(0)} \exp[-\theta c_{kk'}]\, z_{k'n}
\end{cases} \qquad (1.38)
$$

These Equations (1.37 and 1.38) also hold for general graphs containing cycles but, in this case, the problem is no more multi-level, as for an acyclic lattice. Therefore, for general graphs with cycles, the Equations (1.37 and 1.38) define two systems of linear equations that have to be solved (as shown by Saerens et al. (2009) by using a statistical physics framework). Although the idea of calculating the partition function in terms of recurrence relations is not new (see e.g., Zhang and Marr (1995)) and is already computed with dynamic programming (see e.g., Newberg and Lawrence (2009)), the above recurrence relations are valid not only for lattices, but also for any kind of graph. Notice that if the destination node is not turned into an absorbing node, non-hitting paths are considered and Equation (1.38) should be rewritten as

$$z_{nn} = 1 + \sum_{k' \in \mathrm{Succ}(n)} p_{nk'}^{(0)} \exp[-\theta c_{nk'}]\, z_{k'n} \tag{1.39}$$

Interestingly, these recurrence formulae are quite similar to the well-known forward-backward procedure appearing in hidden Markov models (HMM, see Rabiner (1990); Rabiner and Juang (1993)). In fact, it can be shown that by assuming $\theta = 1$, defining $c_{kk'} = 0$ and extending the lattice with "emission" nodes (modeling emission probabilities), the Sum-over-Paths formalism reduces to the forward/backward recurrences of the Baum-Welch algorithm for training hidden Markov models.

Chapter 2

The Sum-over-Paths edit distances

Contents

Determining the similarity of two sequences is a central subject in many fields, such as pattern recognition (Theodoridis and Koutroumbas, 2006), computer science (Cormen et al., 2000; Stephen, 1994), bioinformatics (Durbin et al., 1998; Kruskal, 1983; Sankoff and Kruskal, 1983), and computational linguistics (Jurafsky and Martin, 2000). By applying this similarity, one can determine the class of a new observed sequence from the already known data, e.g., in bioinformatics we can infer the type of protein encoded in a DNA sequence by comparing it with the existing protein data.

Approximate String Matching (ASM) techniques compute this similarity as the best, optimal, alignment between two sequences. By optimal, we mean

Figure 2.1: Two nodes connected by one single shortest path, A and B, are considered less similar than two nodes connected by multiple sub-optimal paths, C and D.

the alignment corresponding to the least total cost, which can be computed by the well-known Viterbi algorithm (Fornay, 1973). The alignment can be seen as a path in the editing graph which transforms one sequence into another.

The aim of this chapter is to extend, in a straightforward way, the basic ASM algorithms in the following way: instead of only relying on the best alignments or paths, we propose to average the total cost over all the possible alignments, the sum over all paths (SoP).

The rationale behind this method, is that we consider that one single optimal path (see Figure 2.1, points A and B) may have been a result of a noisy measure. On the other hand, by having many sub-optimal paths (see Figure 2.1, points C and D), there is a higher chance that that these links are relevant and are not just caused by a noisy observation. By relying on multiple sub-optimal paths, we increase the robustness of the measure, and remove possible noise. We therefore consider randomness useful when some noise is present in the data set.

This is indeed our **research question** for this chapter: to investigate whether a robust distance measure can be obtained by extending the RSP framework, and its relative performance regarding other state of the art distances.

The new measures exploit the paths on a graph, and empirically prove to outperform the basic measures in terms of classification accuracy. We apply the RSP framework by considering all editing paths (ties) between the start node and end node of the editing graph. Strings having many sub-optimal editing paths that transform one into another will be considered, therefore, more similar than strings having a single optimal editing path.

Main contributions

In this chapter we introduce four novel methods based on the Sum-over-Paths framework for Approximate String Matching: (i) the Sum-over-Paths edit distance, (ii) the Sum-over-Paths common subsequences, (iii) the Normalized Sum-over-Paths edit distance, and (iv) the Normalized Sum-over-Paths common subsequences. These measures provide a model-independent technique for computing similarities by taking all alignments into account. They also avoid noisy measures by favoring relevant subsequences of nearly-optimal alignments. Furthermore, their normalized versions overcome any normalization issue.

2.1 Basic notions

Let us first introduce the basic notions upon which the rest of the chapter is based. The notion of graph as a lattice and the editing path are presented in this section. Eventually, we introduce the properties of distances.

2.1.1 Strings, editing operations, and graphs

A *string* $\mathbf{s} = s_1 s_2 ... s_n$ of length n is a concatenation of n symbols, s_i. A *substring* of $\mathbf{s}$ is a string obtained by removing a series of adjacent symbols (prefix or suffix) from the original string, e.g., "car" is a substring of the word "carrot". When the removed symbols are not adjacent we call it a *subsequence*, e.g., "cot" is a subsequence of the word "carrot".

One way of comparing two strings is to transform the first string into the second one, and count the minimum number of operations needed. An example from Sankoff and Kruskal (1983) of transforming string $\mathbf{s}$ = INDUSTRY into string $\mathbf{t}$ = INTEREST, is shown in Figure 2.2. This figure shows the *trace* from $\mathbf{s}$ to $\mathbf{t}$ which contains a series of lines (editing operations) that provide

Operation	**Transformation**	**Cost**
Insert	A new symbol s_i is added to $\mathbf{s}$	$c_{ins} > 0$
Delete	A symbol s_i is removed from $\mathbf{s}$	$c_{del} > 0$
Substitution	A symbol s_i is replaced by s_j in $\mathbf{s}$	$c_{repl} = \begin{cases} > 0 & \text{if } s_i \neq s_j \\ 0 & \text{if } s_i = s_j \end{cases}$

Table 2.1: Editing operations and their cost.

a correspondence, often partial, between both strings. The *editing operations* are namely: inserting, deleting, or replacing a symbol from $\mathbf{s}$ by a symbol from $\mathbf{t}$ (see a summary in Table 2.1).

Each editing operation has a non-negative associated cost (c_{ins}, c_{del}, c_{repl}), which can vary depending on the application. Positive values are usually assigned to c_{ins} and c_{del}, while c_{repl} takes a positive value for $i \neq j$ and zero otherwise. We call an **editing path** the sequence of editing operations that transform a sequence into another.

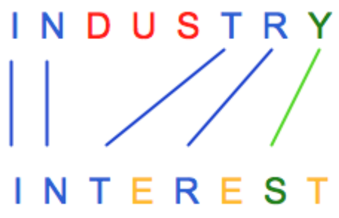


Figure 2.2: Trace of the transformation of string "INDUSTRY" into string "INTEREST". Editing operations are marked in blue (matching substitution), red (delete), orange (insert), and green (non-matching substitution).

This same alignment can be seen as an *editing path* composed of several editing operations on an *edit graph*. An *edit graph* is a lattice of dimensions $|\mathbf{s}| \times |\mathbf{t}|$, where each transition corresponds to an editing operation between two symbols (arrow) and no previous state can be reached (no backward arrows). Figure 2.3 shows the same alignment as an editing path on an edit graph. Please note that every state of the edit graph can only be reached from three previous positions as shown in Figure 2.4. We therefore know the set of predecessor states ($\text{Pred}(i,j) = \{(i-1,j),(i,j-1),(i-1,j-1)\}$), and

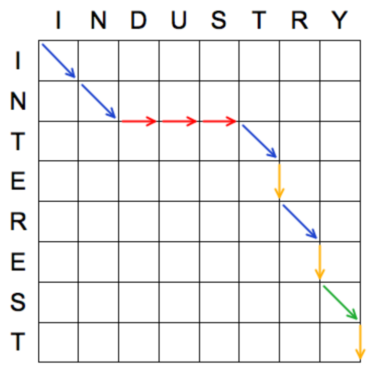


Figure 2.3: Editing path of the transformation of string “INDUSTRY” into string “INTEREST”. Editing operations are marked in blue (matching substitution), red (delete), orange (insert), and green (non-matching substitution).

successor states ($\text{Succ}(i,j) = \{(i+1,j),(i,j+1),(i+1,j+1)\}$).

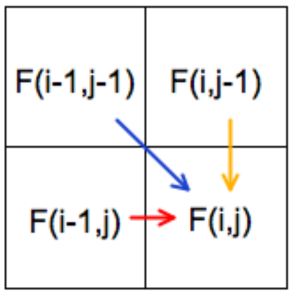


Figure 2.4: State (i,j) can be reached from previous states $(i-1,j)$, $(i,j-1)$, and $(i-1,j-1)$ by a deletion, insertion, or substitution, respectively. Therefore, the value of $F(i,j)$ can be computed as a combination of the three previous ones.

We can now define a basic *distance between two strings* as the sum of the individual costs that compose the editing path. Obviously, the presented editing path is not the only possible path. Figure 2.5 shows two alternative editing paths to transform string “WATER” into string “WINE”. Let us now assume that editing costs, c_{ins}, c_{del}, c_{repl}, have a unit value. In this case, path (a) has a cost of 4 (matching substitutions in blue have a 0 cost), while path (b) has a cost of 5. The problem that arises is how to evaluate the different paths (alignments), and which option is the most accurate one to define the similarity

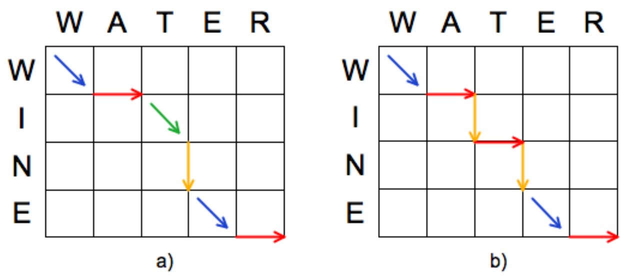


Figure 2.5: Two possible editing paths to transform string “WATER” into string “WINE”. Editing operations are marked in blue (matching substitution), red (delete), orange (insert), and green (non-matching substitution).

or distance between any two strings. The following sections introduce two standard methods: the edit distance and the longest common subsequence similarity.

2.1.2 The standard edit distance

In the previous section we have seen how two sequences may be aligned by different editing paths. However, for every two strings, there is an edit path that involves a minimum number (or cost) of operations, therefore the minimal distance, which is called the *edit distance* between two strings. Thus, the edit distance is the cheapest sequence of edit operations that transform the first sequence into the second one. When the costs for editing operations are equal to the unity, the edit distance is called the *Levenshtein edit distance* (LED)[1]. The edit distance can be computed thanks to a dynamic programming framework (see for instance Durbin et al. (1998); Gusfield (1997); Navarro (2001); Stephen (1994); Wagner and Fischer (1974)).

This dynamic programming algorithm occurs on a lattice of dimensions $(|\mathbf{s}|+1)\times(|\mathbf{t}|+1)$ where each node $k = (i, j)$ corresponds to a state of the editing procedure (each cell from Figures 2.4 and 2.5 is a state). In the general case of an acyclic directed graph, we have to consider that the states (or nodes) of the

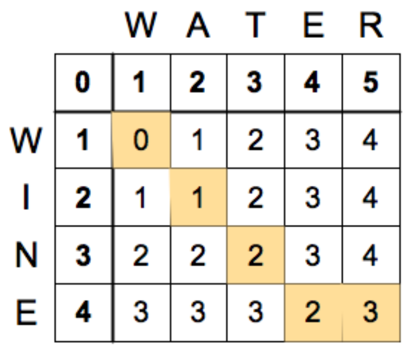
[1]Note that unit costs are assumed for all editing operations, unlike the edit distance where different costs can be used. We use both terms as synonyms throughout the document, but unit costs are always assumed for the LED.

editing graph have been reordered in such a way that node 1 is the starting node and node n is the ending node. Moreover, a topological ordering has been performed. The resulting directed graph contains $n = (|\mathbf{s}|+1) \times (|\mathbf{t}|+1)$ nodes and is an acyclic lattice. It is assumed that this lattice has been pre-computed and that, for each state $k = (i, j)$, the sets $\text{Pred}(k)$ of predecessor nodes and successor nodes $\text{Succ}(k)$ are available.

The dynamic programming algorithm computes the values of the edit distance, $F(i, j)$, for each intermediate state, (i, j) with $1 \leq i \leq (|\mathbf{s}| + 1)$ and $1 \leq j \leq (|\mathbf{t}| + 1)$, with the following equation:

$$F(i,j) = \begin{cases} 0 & \text{for } F(1,1) \\ min \begin{cases} F(i-1, j-1) + c_{repl} \\ F(i, j-1) + c_{ins} \\ F(i-1, j) + c_{del} \end{cases} & \text{otherwise} \end{cases} \tag{2.1}$$

This equation can be interpreted as the minimum number of operations up to the current state. An example of the application of this procedure can be found in Figure 2.6. The edit distance is, therefore, the value of $F(|\mathbf{s}|+1, |\mathbf{t}|+1)$; in the example this value is equal to three editing operations, $F(5, 6) = 3$. This



		W	A	T	E	R
	0	1	2	3	4	5
W	1	0	1	2	3	4
I	2	1	1	2	3	4
N	3	2	2	2	3	4
E	4	3	3	3	2	3

Figure 2.6: Dynamic programming algorithm for calculation of the edit distance between strings “WATER” and “WINE”. The shortest editing path is marked in orange.

standard formulation only considers the optimal alignment between the two sequences, and can be viewed as a kind of Viterbi algorithm (Fornay, 1973; Viterbi, 1967). Furthermore, the edit distance is a *distance metric* (Vitányi, 2011), as it complies with the properties from Table 2.2.

Non-negative property:	$d(\mathbf{s},\mathbf{t})$	$\geq$	$0\ \forall \mathbf{s},\mathbf{t}$
Zero property:	$d(\mathbf{s},\mathbf{t})$	$=$	$0 \iff \mathbf{s}=\mathbf{t}$
Symmetry property:	$d(\mathbf{s},\mathbf{t})$	$=$	$d(\mathbf{t},\mathbf{s})\ \forall \mathbf{s},\mathbf{t}$
Triangle inequality property:	$d(\mathbf{s},\mathbf{t})$	$\leq$	$d(\mathbf{s},\mathbf{r})+d(\mathbf{r},\mathbf{t})\ \forall \mathbf{s},\mathbf{t},\mathbf{r}$

Table 2.2: Distance metric properties.

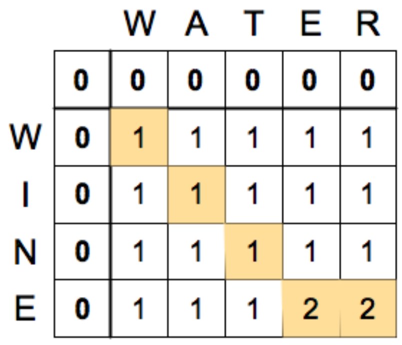
2.1.3 The standard longest common subsequence

Another frequently used measure between two strings is the longest common subsequence similarity. The Longest Common Subsequence (LCS) algorithm (Cormen et al., 2000; Stephen, 1994) computes this similarity (together with the longest common subsequence itself, if desired) through a method similar to edit distance computation. The dynamic programming method is computed as follows:

$$F(i,j) = \begin{cases} 0 & \text{for } F(1,1) \\ max \begin{cases} F(i-1,j-1)+s_{i,j} \\ F(i,j-1) \\ F(i-1,j) \end{cases} & \text{otherwise} \end{cases} \tag{2.2}$$

where $s_{i,j}$ is the similarity between states i and j, and is equal, in the standard case, to 1 when $i = j$ and 0 otherwise. An example of the calculation of the LCS for two sequences is shown in Figure 2.7. In this case, the value of the longest common subsequence is 2, as only the subsequence "WE" is shared. Because this measure is a similarity it only complies with the non-negative and symmetric properties from Table 2.2.

Myers (1986) show that the longest common subsequence and the edit distance are dual problems, where the first tries to find the editing path with



		W	A	T	E	R
	0	0	0	0	0	0
W	0	1	1	1	1	1
I	0	1	1	1	1	1
N	0	1	1	1	1	1
E	0	1	1	1	2	2

Figure 2.7: Dynamic programming algorithm for calculation of the longest common subsequence between strings “WATER” and “WINE”. The shortest editing path is marked in orange.

maximum number of diagonal edges (matching substitutions), while the second focuses on editing paths with minimum number non-diagonal edges (insertions and deletions).

The relation between the LCS and the ED for two strings **s** and **t** is: ED $= |\mathbf{s}| + |\mathbf{t}| - 2 \times$ LCS. Please note that in this case, we use the Edit Distance, with weights for replacement of 2 (a replacement is the sum of an insertion and a deletion). In this case, the ED for the strings “Wine” and “Water” is 5. We can clearly see that the relation between the ED and the LCS holds: $5 = 4 + 5 - 2 \times 2$.

2.2 Related work

In the previous sections we have introduced the standard methods for finding the optimal, shortest, alignments between two sequences. However, as shown in Figure 2.5, there may exist several possible alignments between two sequences. Given that there are alternative alignments with nearly the same cost as the optimal one, one may want to consider also these sub-optimal alignments. Indeed, as noticed by (Durbin et al., 1998), relevant information is also contained in these sub-optimal alignments. For some applications, *two sequences sharing many sub-optimal alignments should be considered as more similar than two* *sequences having only one single, optimal, alignment in common.* To this end, we adopt a Sum-over-Paths (SoP) formalism, considering that each alignment corresponds to a path on the dynamic programming lattice.

Within the context of computing edit distances between sequences, there have been successful attempts to account for all possible alignments (see, e.g., Bucher and Hofmann (1996); Durbin et al. (1998); Krogh et al. (1994); Leslie et al. (2004); Lodhi et al. (2002); Rousu and Shawe-Taylor (2005); Shawe-Taylor and Cristianini (2004); Watkins (1999), among others). The sequence comparison model introduced by Durbin et al. (1998) is quite popular in bioinformatics. It is based on a hidden Markov model (HMM) of sequence pairs generation, called the pair HMM. The model is trained by maximum likelihood on a sequences sample and, once is trained, it provides the likelihood of observing any two sequences, each alignment being weighted by its probability.

The SoP edit distance introduced in this chapter shares therefore some similarities with the pair HMM, but is much simpler since it does not require any transition-probability estimation – it is model-free, although by modifying the edition costs or the similarity measure we could adapt it to any model or specific task.

It is, however, also possible to fix a priori the transition probabilities of the pair HMM. In that case, the SoP edit distance is equivalent to the pair HMM with a suitable choice of the editing costs, and parameter θ equal to 1. However, two important differences between the proposed SoP techniques and pair HMMs is that the SoP edit distance

1. weights the contribution of the different alignments, according to their respective total costs, and

2. depends on a parameter allowing to regulate the degree of exploration and thus the contribution of sub-optimal paths.

It therefore encompasses both the Viterbi and the Baum-Welch algorithms as special cases. It will be observed in the experimental section that the best performance obtained by the SoP techniques is often achieved for θ parameter values greater than 1. Finally, notice that a valid kernel derived from a pair HMM was proposed by Jaakkola et al. (2000) and Watkins (1999).

The string kernels introduced by Leslie et al. (2004); Liao and Noble (2002); Lodhi et al. (2002); Rousu and Shawe-Taylor (2005); Saigo et al. (2004); Shawe-Taylor and Cristianini (2004) compute a score depending on all the possible common subsequences of the two compared sequences. The main idea behind these sequence-comparison techniques is to reward common subsequences – contiguous or not, depending on the technique. However, when the size of the alphabet is very low, there is a huge quantity of such short common subsequences and very few long common subsequences, so that the obtained score does not reflect accurately the proper similarity between the two sequences. Indeed, a much larger weight should be put on long common subsequences; this is exactly what the SoP edit distance does. We therefore expect string kernels to perform well when the alphabet is quite large (for instance in the context of text mining – in this case, there are very few long common subsequences; see for example Cancedda et al. (2003)), and worse in the case of reduced alphabets – this is indeed observed in our experiments.

Yet another approach computing an edit score along all possible alignments is the stochastic edit distance, and related methods (Bahl and Jelinek, 1975). The underlying model is based on a noisy channel. An input string $\mathbf{s}$ – the reference string or code – is distorted by a noisy channel, therefore producing an output string $\mathbf{t}$ that is a noisy transform of $\mathbf{s}$. The noisy channel is often modeled as a Markov model or a transducer (Bahl and Jelinek, 1975; Oncina and Sebban, 2006). For these models, the probability of generating string $\mathbf{t}$ from $\mathbf{s}$, $\mathrm{P}(\mathbf{t}|\mathbf{s})$, can be computed thanks to a forward recursion formula involving all possible ways of generating $\mathbf{t}$, and therefore all possible paths through the lattice. The scores $-\log \mathrm{P}(\mathbf{t}|\mathbf{s})$ or $-\log \mathrm{P}(\mathbf{s}|\mathbf{t})$ can be considered as dissimilarity measures between $\mathbf{s}$ and $\mathbf{t}$. This stochastic edit distance model is refined in Ristad and Yianilos (1998b); Amengual and Vidal (2000); Oncina and Sebban (2006), where the local distances between the symbols are estimated from sample data. A paragraph explaining the relationships between stochastic edit distances and the sum-over-paths approach appears at the end of Section 2.3.1.

However, these stochastic edit distances do not have a parameter biasing the measure towards minimum cost solutions.

Path randomization applications are also found within the bioinformatics literature. A stochastic alignment procedure is presented by Thorne et al. (1991), where the authors introduce an evolutionary model based on transition probabilities which allow not only to estimate the optimal alignment via the maximum likelihood, but also to estimate the set of evolutionary parameters given two aligned sequences. This work is further extended by Thorne et al. (1992) by adding the possibility of multiple-base insertion and deletion as well as heterogeneous evolutionary rates. This evolutionary model is also extended by Steel and Hein (2001) by generalizing the pairwise recursions to an r-sequence star-shaped tree.

Zhang and Marr (1995) propose a method which extends the typical dynamic programming method by allowing uncertainty and, thus, weighting the fluctuating sub-optimal alignments. This probability distribution has its basis on the partition function, which is computed via a recurrence formula. The average cost is also computed, though is not used as similarity measure between two sequences. Furthermore, this model is restricted to lattices and there are no methodic experimental results, although simulation experiences are suggested.

A formulation for the alignments with highest probability, according to a Boltzmann distribution based on the partition function and a temperature parameter, is provided by Miyazawa (1995). The author defines a pairwise similarity measure consisting on a log-odds matrix of amino acid mutations, which is then used to compute the statistical weight of an alignment. Moreover, a formulation of pairwise probabilities in terms of forward and backward variables is presented.

Eventually, an alignment consisting of the most probable pairwise correspondences is provided. Hwa and Lässig (1996) introduce a similarity detection method based on statistical physics. The idea behind this paper is that the large-scale statistics of the fluctuating paths can be derived from the partition function in a path integral representation. Other similar probability alignments are also presented by Kschischo and Lässig (2000), where the finite-temperature alignments are introduced. These stochastic alignments are derived from a thermodynamic partition function and they can be used for assessing the relevance of sub-optimal paths. The authors define the weight of each alignment proportionally to the exponential of a defined similarity measure which depends on the number of matches, mismatches and gaps. Eventually, the probability of a pairwise element is computed as the sum of the probabilities over all paths that contain the pair. This partition function formalism is also computed by Newberg and Lawrence (2009) via a dynamic programming algorithm, where the authors introduce a probabilistic approach to calculate the alignment score based on a Boltzmann distribution.

However, and to the best of our knowledge, there is no work that uses the average cost of the fluctuating sub-optimal paths as a similarity measure between two sequences to avoid noisy measures, yet many of them agree that the optimal alignment may not be the best one, and that other sub-optimal paths are also relevant. The present work introduces a SoP formalism which allows to compute relevant quantities on a graph, such as the expected cost or the expected number of passages through a node. A probability distribution is assigned on the set of paths and the quantities of interest are computed with respect to this distribution. The derivation of these quantities differs from the one proposed by Mantrach et al. (2010); Saerens et al. (2009); Yen et al. (2008) in that it is more intuitive, in our opinion, as it directly derives the main quantities without having to compute the derivatives of a partition function. Within this formalism, extensions of the edit distance and the longest common subsequence are developed, taking all the alignments into account.

It must also be stressed that these previous works focus only on acyclic lattices, while the presented approach is generalized to arbitrary graphs (including cycles). Furthermore, although the use of the partition function formalism proves to be useful in this context, any of the previous work derives it from an optimization problem (as is the case of the Sum-over-Paths edit distances). Eventually, previous work is mostly focused on biologically relevant similarity or cost functions, while generic edit distance costs are applied here, leading to an application-independent method which proves to be useful in OCR tasks.

Finally, let us also mention that some authors (Do et al., 2006; Oncina and Sebban, 2006; Ricci et al., 2007; Ristad and Yianilos, 1998a) propose to tune the edit costs from a training set, which results in better performances at the cost of increasing computational complexity. Moreover, an edit graph (lattice) for a pair of sequences can be thought of as a strong product graph of two chain graphs each representing one of the two given sequences, the SoP approach is also related to graph kernels working on product graphs (Vishwanathan et al., 2010, 2007). For instance, Vishwanathan et al. (2010) framework of graph kernels viewed as path counting in product graphs could probably be adapted to strong product graphs and thus applied to sequence comparison.

2.3 The Sum-over-Paths edit distance and common subsequence

2.3.1 A Sum-over-Paths extension of the edit distance

Let us now turn to our Sum-over-Paths (SoP) formalism, computing the distance along all the alignments between the two sequences, which is a direct application of the randomized shortest-path formalism introduced in Chapter 1.

The RSP framework provides a set of distances that lie between (i) the *Commute-Time distance* (Gobel and Jagers, 1974), which is defined as the expected length of paths that a random walker can follow between a pair of nodes, and (ii) the *Shortest Path distance*, which is defined as the length of the shortest path between a pair of nodes. The advantage of using multiple paths, unlike the shortest path distance, has already been demonstrated by several authors (Yen et al., 2008), as it provides more robust distance measures for string comparison.

On the other hand, when the graph becomes large enough, the Commute-Time distance is affected by the stationary distribution of the natural random walk on the graph (Brand, 2005). The reason is that when the number of edges connecting a node increases significantly, the random walker has too many paths to follow, and the probability of reaching the destination node becomes dependent on the number of edges. This motivates the RSP distances, which favor sub-optimal paths and provide a robust, yet accurate, distance measure.

As introduced in Section 2.1.2, the edit distance only considers the optimal alignment between two sequences. In order to apply the SoP framework to the edit distance to build a more robust measure, a probability is assigned to each path depending on its optimality or cost.

2.3.2 The SoP edit distance

Let us suppose a 40×40 lattice as the one from Figure 2.8, where each state is represented by a pixel, and the probability of a certain state is given by a color scale, i.e., red for probability equal to one and blue for zero probability. Figure 2.8 (a) shows the probabilities of the edit distance, where the optimal path is marked in red (probability equal to one for each state of the optimal path), while probabilities in Figure 2.8 (b) are distributed among all sub-optimal paths. This kind of distribution is the one that we are interested in, as it favors the optimal path, while still taking into account the sub-optimal ones.

In a more formal way, we can consider the set of all possible paths $\mathcal{P}_{1n}$ between the starting node 1 (top left node in Figure 2.8) and the ending node n (bottom right node in Figure 2.8) on the dynamic programming lattice. Each of these paths $\wp \in \mathcal{P}_{1n}$ corresponds to a possible alignment between sequence $\mathbf{s}$ and sequence $\mathbf{t}$. The probability of occurrence of each alignment is assumed to follow a **Boltzmann distribution** (see Equation (1.6)) as shown in Figure 2.8. Note that here we consider the initial policy $\mathrm{P}^{(0)}$ equally distributed, therefore the Boltzmann distribution can be rewritten as

$$\mathrm{P}(\wp) = \frac{\exp[-\theta C(\wp)]}{\mathcal{Z}} \tag{2.3}$$

where $C(\wp)$ is the total cost associated to the path or alignment $\wp$, that is,

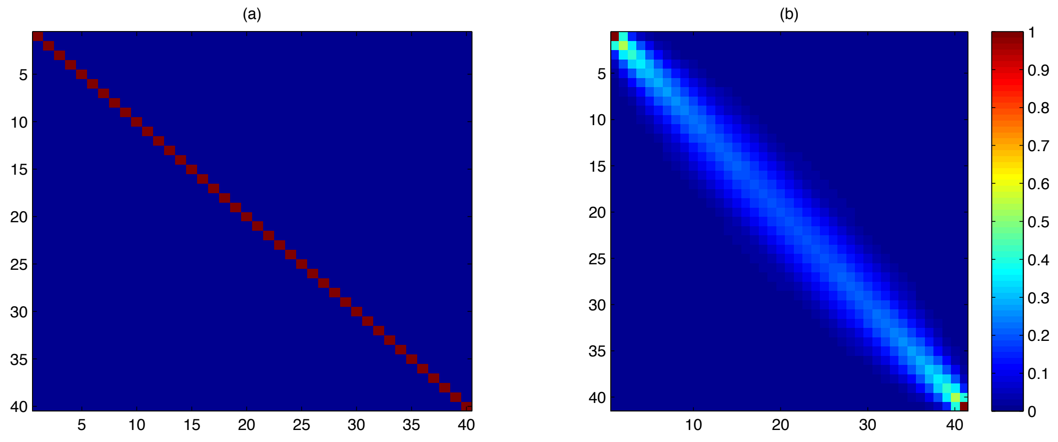


Figure 2.8: Probability of passing through a state on a 40 × 40 lattice. Each state (pixel) has a probability indicated by the color scale on the right. Figure (a) shows the optimal alignment of the edit distance, and (b) the SoP edit distance with $\theta = 1$.

the sum of the individual costs $c_{kk'}$ occurring on that path $\wp$. Please note the normalization factor, or partition function, $\mathcal{Z}$ is given in Equation (2.4). Notice that this partition function has already been used for sequence alignment with uncertainty (see i.e., Kschischo and Lässig (2000); Miyazawa (1995); Zhang and Marr (1995)), for similarity detection as by Hwa and Lässig (1996), or in alignment scoring as by Newberg and Lawrence (2009).

$$\mathcal{Z} = \sum_{\wp \in \mathcal{P}_{1n}} \exp[-\theta C(\wp)] \tag{2.4}$$

In fact, the probability distribution defined in Equation (2.3) minimizes the expected cost subject to a Shannon entropy constraint,

$$\left| \begin{array}{lll} \underset{\mathrm{P}(\wp)}{\text{minimize}} & \sum_{\wp \in \mathcal{P}_{1n}} \mathrm{P}(\wp) C(\wp) & \\ \text{subject to} & -\sum_{\wp \in \mathcal{P}_{1n}} \mathrm{P}(\wp) \ln \mathrm{P}(\wp) & = \; H_0 \end{array} \right. \tag{2.5}$$

where H_0 is the predefined entropy and is inversely related to θ (the lower θ, the larger the entropy), the *inverse temperature* parameter. In other words, it guarantees a minimum expected cost for reaching node n from node 1 while fixing the level of entropy (exploration) spread in the lattice. Thus, the entropy is used instead of relative entropy in Equation (1.5).

It is clear that this definition of the probability distribution on the set of paths favors good alignments (with a low cost $C(\wp)$) over bad ones (with a large cost $C(\wp)$). The parameter θ regulates the sharpness of the distribution: when $\theta \to \infty$, only the best alignments matter while when $\theta \to 0$, all alignments have almost the same probability mass. In other words, the larger the value of parameter θ, the less the impact of sub-optimal paths.

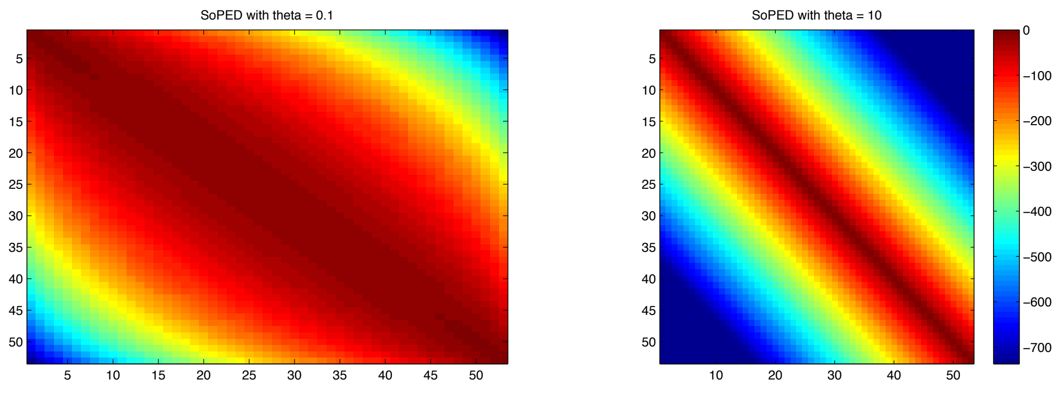


Figure 2.9: Effect of θ on the probabilities of paths with the SoP edit distance. Big values of θ provide a more peaked probability distribution on the optimal alignment (the diagonal path in this case), while for smaller values the distribution is more spread over the lattice. The logarithm of the probabilities is represented here.

Now, the total expected cost for reaching node n from node 1, defining the **SoP edit distance** d_{SoP} between the two sequences, is given by

$$d_{\text{SoP}} = \mathbb{E}[C] = \sum_{\wp \in \mathcal{P}_{1n}} \mathrm{P}(\wp)\, C(\wp) = \sum_{\wp \in \mathcal{P}_{1n}} C(\wp)\, \frac{\exp[-\theta C(\wp)]}{\mathcal{Z}} \tag{2.6}$$

By taking into account the definition of the total cost from Equation (1.25), we can rewrite Equation (2.6) as

$$
\begin{aligned}
d_{\text{SoP}} &= \sum_{\wp \in \mathcal{P}_{1n}} C(\wp) \, \frac{\exp[-\theta C(\wp)]}{\mathcal{Z}} && (2.7)\\
&= \sum_{\wp \in \mathcal{P}_{1n}} \left(\sum_{k=1}^{n} \sum_{k'=1}^{n} \delta(\wp; k, k') \, c_{kk'} \right) \frac{\exp[-\theta C(\wp)]}{\mathcal{Z}} && (2.8)\\
&= \sum_{k=1}^{n} \sum_{k'=1}^{n} c_{kk'} \left(\sum_{\wp \in \mathcal{P}_{1n}} \delta(\wp; k, k') \, \frac{\exp[-\theta C(\wp)]}{\mathcal{Z}} \right) && (2.9)\\
&= \sum_{k=1}^{n} \sum_{k'=1}^{n} c_{kk'} n_{kk'} && (2.10)
\end{aligned}
$$

where we used the definition provided by Equation (1.24). Thus, the SoP edit distance is just the sum of the individual costs multiplied by the expected number of passages through the corresponding links.

Finally, by integrating the definition of the expected number of passages in terms of the forward and backward variables (see Equation (1.34)), Equation (2.10) can be rewritten as

$$
\begin{aligned}
d_{\text{SoP}} &= \frac{1}{z_{1n}} \sum_{k=1}^{n} \sum_{k'=1}^{n} z_{1k} c_{kk'} \exp[-\theta c_{kk'}] z_{k'n} && (2.11)\\
&= \frac{1}{z_{1n}} \sum_{k=1}^{n} \sum_{k' \in \text{Succ}(k)} z_{1k} c_{kk'} \exp[-\theta c_{kk'}] z_{k'n} && (2.12)
\end{aligned}
$$

This will allow us computing the SoP edit distance in terms of the immediate costs and the forward/backward variables.

2.3.3 Link with the stochastic edit distance

As noted by a reviewer of García-Díez et al. (2011), the SoP model is also closely related to the stochastic edit distance (SED) by Bahl and Jelinek (1975). Indeed, the path probability proposed is approximately the same as the conventional likelihood of an alignment path $\mathrm{P}(\mathbf{y}|\mathbf{x})$ between a given string, $\mathbf{x}$, and its "distorted version", $\mathbf{y}$, under the assumption that the (insertion, deletion, substitution) error probabilities are uniform.

More precisely, let p_e be the probability of each of the (insertion, deletion, non-matching substitution) errors and p_m the probability for a matching substitution. The likelihood of $\mathbf{y}$ along path $\wp$ is the product of the probabilities of all the edit operations along $\wp$, i.e.,

$$\mathrm{P}(\mathbf{y}, \wp|\mathbf{x}) = (p_e)^{n_e}\,(p_m)^{n_m} \tag{2.13}$$

where n_e is the number of errors and n_m is the number of matches along $\wp$. If we define the related costs

$$c_e = -\log p_e \tag{2.14}$$

and

$$c_m = -\log p_m \tag{2.15}$$

the $\mathrm{P}(\mathbf{y}, \wp|\mathbf{x})$ can be rewritten as

$$\mathrm{P}(\mathbf{y}, \wp|\mathbf{x}) = \exp\left[-\sum_{k\to k'\in\wp} c_{kk'}\right] = \exp[-C(\wp)] \tag{2.16}$$

which turns out to be the same form as the SoP path probability with $\theta = 1$. In that case,

$$\mathrm{P}(\mathbf{y}|\mathbf{x}) = \sum_{\wp\in\mathcal{P}_{1n}} \mathrm{P}(\mathbf{y}, \wp|\mathbf{x}) = \sum_{\wp\in\mathcal{P}_{1n}} \exp[-C(\wp)] \tag{2.17}$$

which corresponds to the partition function, $\mathcal{Z}$. Therefore, in this context, the partition function can be considered as a likelihood function, and $-\log\mathcal{Z}$ a dissimilarity measure between string $\mathbf{x}$ and string $\mathbf{y}$. This SED dissimilarity measure is investigated in the experimental section.

2.3.4 A Sum-over-Paths extension of the longest common subsequence

Let us now turn to the randomized version of the longest common subsequence (LCS). The dynamic programming lattice used by the LCS algorithm is in fact identical to that of the Levenshtein edit distance, and exactly the same SoP-based development can be applied, with only one (but important) difference: since LCS is a similarity measure, it involves immediate rewards or similarities $s_{kk'}$ instead of costs. Therefore, the probability distribution on the set of alignments is redefined as $\mathrm{P}(\wp) = \exp[\theta S(\wp)]/\mathcal{Z}$ where $S(\wp)$ is the total similarity associated to path $\wp$ (the sum of the individual similarities along the path) and $\mathcal{Z} = \sum_{\wp \in \mathcal{P}_{1n}} \exp[\theta S(\wp)]$. Thus, good alignments (having a large similarity) are favored over bad ones (having a low similarity).

The **SoP common subsequence similarity**, denoted as s_{SoP}, is defined as the average of $S(\wp)$ over the previous probability distribution. By proceeding exactly as in the previous section, we obtain

$$s_{\mathrm{SoP}} = \frac{1}{z_{1n}} \sum_{k=1}^{n} \sum_{k'=1}^{n} z_{1k} s_{kk'} \exp[\theta s_{kk'}] z_{k'n} \tag{2.18}$$

Notice that in the SoP CS case, as opposed to the SoP ED, cycles are not allowed, since the similarity could become arbitrarily high by looping.

2.3.5 Gap handling with SoP edit distances

It must be noticed that our model already handles gaps, i.e., contiguous non-matching subsequences, by allowing insertions and deletions, yet one may like to handle affine gaps (Durbin et al., 1998). Let us imagine that we have a lattice as the one in Figure 2.10, where each node is represented twice to model the possibilities of continuing a gap, or starting a gap.

The new graph has an increased number of edges in order to allow different costs depending whether we have already started the gap, or we start a new one (as we prefer fewer long gaps over many short ones). The number of

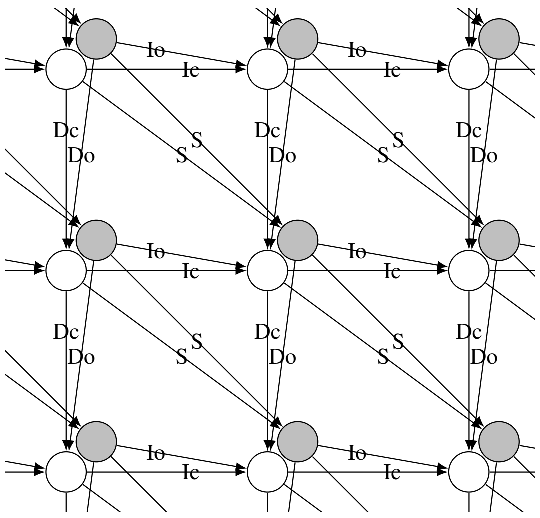


Figure 2.10: Graph that allows affine gaps: the white nodes represent the continuation of a gap while grey nodes represent the start of a gap. DO/IO edges start a new gap, while DC/IC edges continue a gap.

possible paths remains the same as the possible editing operations are the same from any node, although with different costs. Furthermore, the expected cost remains the same when a linear gap model cost is assumed. For different affine gap models, from the linear model, the local costs would change letting the probability distribution vary and, therefore, the expected cost as well. It would be thus possible to integrate affine gaps in our solution and still use the same recurrence relations for computing the expected cost.

2.3.6 The SoP edit distance algorithm

The SoP edit distance can thus be computed as follows (see Algorithm 1):

1. Compute the forward and backward variables using Equations (1.37) and (1.38).
2. Compute the SoP edit distance thanks to Equation (2.12).

The code of the SoP edit distance algorithm is available at `http://github.com/silviagdiez/thesis`. This algorithm depends on the parameter θ and assumes constant insertion and deletion costs, c_{ins} and c_{del}.

The states of the dynamic programming lattice are indexed by $k := (i, j)$. The insertion of a symbol corresponds to a transition $(i, j) \rightarrow (i, j+1)$ while the deletion of a symbol corresponds to $(i, j) \rightarrow (i+1, j)$. A substitution is $(i, j) \rightarrow (i+1, j+1)$ with cost $c_{\text{sub}}(s_1(i), s_2(j))$. The first line ($j = 1$) and column ($i = 1$) correspond to a dummy, empty, symbol in the dynamic programming table. The forward and backward variables are denoted by z_{f} (for the z_{1k}) and z_{b} (for the z_{kn}), respectively.

Algorithm 1 SumOverPathsED: Computation of the SoP Edit Distance.

```
Input:
   • $\theta > 0$: the parameter controlling the degree of randomness.
   • $\mathbf{s}_1$, $\mathbf{s}_2$: sequences of symbols of length $n_1$, $n_2$ respectively.
   • $c_{\text{ins}}$, $c_{\text{del}}$, $c_{\text{sub}}(x, y)$: edit costs (insertion, deletion, substitution).
Output:
   • $d_{\text{SoP}}$: the Sum-over-Paths edit distance.
1. Call ForwardRecurrenceED (Algorithm 2)
2. Call BackwardRecurrenceED (Algorithm 3)
3. $\mathcal{Z} = z_{\text{f}}(n_1 + 1, n_2 + 1)$
4. Initialization of the expected cost $d$ from the transitions on the first row
   and the first column of the lattice:
   $d = \sum_{i=1}^{n_1} z_{\text{f}}(i, 1) c_{\text{del}} \exp[-\theta c_{\text{del}}] z_{\text{b}}(i+1, 1)$
      $+ \sum_{j=1}^{n_2} z_{\text{f}}(1, j) c_{\text{ins}} \exp[-\theta c_{\text{ins}}] z_{\text{b}}(1, j+1)$
5. for $i = 1$ to $n_1$ do
6.    for $j = 1$ to $n_2$ do
7.       $d = d + z_{\text{f}}(i, j) c_{\text{sub}}(s_1(i), s_2(j)) \exp[-\theta c_{\text{sub}}(s_1(i), s_2(j))] z_{\text{b}}(i+1, j+1)$
            $+ z_{\text{f}}(i+1, j) c_{\text{ins}} \exp[-\theta c_{\text{ins}}] z_{\text{b}}(i+1, j+1)$
            $+ z_{\text{f}}(i, j+1) c_{\text{del}} \exp[-\theta c_{\text{del}}] z_{\text{b}}(i+1, j+1)$
8.    end for
9. end for
10. return $d_{\text{SoP}} = d/\mathcal{Z}$
```

2.3.7 The SoP common subsequence algorithm

The SoP common subsequences can be computed as follows (see Algorithm 4):

Algorithm 2 ForwardRecurrenceED: Computation of the forward variables for the SoP edit distance.

```
Input:
    • $\theta > 0$: the parameter controlling the degree of randomness.
    • $\mathbf{s}_1$, $\mathbf{s}_2$: sequences of symbols of length $n_1$, $n_2$ respectively.
    • $c_{\text{ins}}$, $c_{\text{del}}$, $c_{\text{sub}}(x, y)$: edit costs (insertion, deletion, substitution).
Output:
    • $z_{\text{f}}$: the forward recurrence table.
1. Initialization of the forward variables $z_{\text{f}}(i, j)$:
   $z_{\text{f}}(i, 1) = (\exp[-\theta c_{\text{del}}])^{(i-1)}$, for $i = 1$ to $(n_1 + 1)$
   $z_{\text{f}}(1, j) = (\exp[-\theta c_{\text{ins}}])^{(j-1)}$, for $j = 1$ to $(n_2 + 1)$
2. for $i = 1$ to $n_1$ do
3.   for $j = 1$ to $n_2$ do
4.     $z_{\text{f}}(i + 1, j + 1) = z_{\text{f}}(i, j) \exp[-\theta c_{\text{sub}}(s_1(i), s_2(j))]$
              $+ z_{\text{f}}(i + 1, j) \exp[-\theta c_{\text{ins}}] + z_{\text{f}}(i, j + 1) \exp[-\theta c_{\text{del}}]$
5.   end for
6. end for
7. return $z_{\text{f}}$
```

Algorithm 3 BackwardRecurrenceED: Computation of the backward variables for the SoP edit distance.

```
Input:
    • $\theta > 0$: the parameter controlling the degree of randomness.
    • $\mathbf{s}_1$, $\mathbf{s}_2$: sequences of symbols of length $n_1$, $n_2$ respectively.
    • $c_{\text{ins}}$, $c_{\text{del}}$, $c_{\text{sub}}(x, y)$: edit costs (insertion, deletion, substitution).
Output:
    • $z_{\text{b}}$: the backward recurrence table.
1. Initialization of the backward variables $z_{\text{b}}(i, j)$:
   $z_{\text{b}}(i, 1) = (\exp[-\theta c_{\text{del}}])^{(n_1+1-i)}$, for $i = (n_1 + 1)$ to 1
   $z_{\text{b}}(1, j) = (\exp[-\theta c_{\text{ins}}])^{(n_2+1-j)}$, for $j = (n_2 + 1)$ to 1
2. for $i = n_1$ downto 1 do
3.   for $j = n_2$ downto 1 do
4.     $z_{\text{b}}(i, j) = z_{\text{b}}(i + 1, j + 1) \exp[-\theta c_{\text{sub}}(s_1(i), s_2(j))]$
              $+ z_{\text{b}}(i, j + 1) \exp[-\theta c_{\text{ins}}] + z_{\text{b}}(i + 1, j) \exp[-\theta c_{\text{del}}]$
5.   end for
6. end for
7. return $z_{\text{b}}$
```

1. Compute the forward and backward variables using Equations (1.37) and (1.38). Please note that here we use the similarities, instead of costs, therefore the initialization and computation of the forward and backward variables is slightly different.

2. Compute the SoP common subsequence thanks to Equation (2.18).

The code of the SoP common subsequence algorithm is available at `http://github.com/silviagdiez/thesis`.

Algorithm 4 SumOverPathsCS: Sum-over-Paths Common Subsequence.

```
Input:
  • $\theta > 0$: the parameter controlling the degree of randomness.
  • $\mathbf{s}_1$, $\mathbf{s}_2$: sequences of symbols of length $n_1$, $n_2$ respectively.
  • $s(x, y)$: symbol similarities.
Output:
  • $s_{\text{SoP}}$: the Sum-over-Paths common subsequence similarity.
1.  Call ForwardRecurrenceCS (Algorithm 5)
2.  Call BackwardRecurrenceCS (Algorithm 6)
3.  $\mathcal{Z} = z_{\text{f}}(n_1 + 1, n_2 + 1)$
4.  Initialization of the expected reward $s$: $s = 0$
5.  for $i = 1$ to $n_1$ do
6.    for $j = 1$ to $n_2$ do
7.      $s = s + z_{\text{f}}(i, j)\ \text{sim}(s_1(i), s_2(j)) z_{\text{b}}(i + 1, j + 1)$,
        where $\text{sim}(x, y) = \begin{cases} \exp[\theta\, s(x, y)] \text{ if } x = y \\ 0 \text{ otherwise} \end{cases}$
8.    end for
9.  end for
10. return $s_{\text{SoP}} = s/\mathcal{Z}$
```

The time and space complexity of both Algorithms 1 and 4 is $O(n_1 n_2)$, where n_1 and n_2 are the length of the two input sequences; i.e., they have the same time complexity as the standard, non-optimized, edit distance computation. However, the space complexity of the edit distance is $O(min(n_1, n_2))$ (by divide and conquer). It must be noticed that the spatial complexity could be improved by applying the *Four Russians* method (Arlazarov et al., 1970), although there is a trade-off between the spatial and time complexity of the algorithm.

Algorithm 5 ForwardRecurrenceCS: Computation of the forward variables for the SoP Common Subsequence.

```
Input:
    • $\theta > 0$: the parameter controlling the degree of randomness.
    • $\mathbf{s}_1$, $\mathbf{s}_2$: sequences of symbols of length $n_1$, $n_2$ respectively.
    • $s(x, y)$: symbol similarities.
Output:
    • $z_{\mathrm{f}}$: the forward recurrence table.
1. Initialization of the forward variables $z_{\mathrm{f}}(i, j)$:
   $z_{\mathrm{f}}(i, 1) = 1$, for $i = 1$ to $(n_1 + 1)$
   $z_{\mathrm{f}}(1, j) = 1$, for $j = 1$ to $(n_2 + 1)$
2. for $i = 1$ to $n_1$ do
3.     for $j = 1$ to $n_2$ do
4.         $z_{\mathrm{f}}(i+1, j+1) = z_{\mathrm{f}}(i, j)\ \mathrm{sim}(s_1(i), s_2(j)) + z_{\mathrm{f}}(i+1, j) + z_{\mathrm{f}}(i, j+1)$,
           where $\mathrm{sim}(x, y) = \begin{cases} \exp[\theta\, s(x, y)] \text{ if } x = y \\ 1 \text{ otherwise} \end{cases}$
5.     end for
6. end for
7. return $z_{\mathrm{f}}$
```

Algorithm 6 BackwardRecurrenceCS: Computation of the backward variables for the SoP Common Subsequence.

```
Input:
    • $\theta > 0$: the parameter controlling the degree of randomness.
    • $\mathbf{s}_1$, $\mathbf{s}_2$: sequences of symbols of length $n_1$, $n_2$ respectively.
    • $s(x, y)$: symbol similarities.
Output:
    • $z_{\mathrm{b}}$: the backward recurrence table.
1. Initialization of the backward variables $z_{\mathrm{b}}(i, j)$:
   $z_{\mathrm{b}}(i, 1) = 1$, for $i = (n_1 + 1)$ to $1$
   $z_{\mathrm{b}}(1, j) = 1$, for $j = (n_2 + 1)$ to $1$
2. for $i = n_1$ downto $1$ do
3.     for $j = n_2$ downto $1$ do
4.         $z_{\mathrm{b}}(i, j) = z_{\mathrm{b}}(i+1, j+1)\ \mathrm{sim}(s_1(i), s_2(j)) + z_{\mathrm{b}}(i, j+1) + z_{\mathrm{b}}(i+1, j)$
           where $\mathrm{sim}(x, y) = \begin{cases} \exp[\theta\, s(x, y)] \text{ if } x = y \\ 1 \text{ otherwise} \end{cases}$
5.     end for
6. end for
7. return $z_{\mathrm{b}}$
```

2.4 Experiments

In this experimental section we aim at:

1. Evaluating the relative performance of the two proposed methods in a classification and clustering task: the SoP ED ($\mathbf{K}_{\text{ED}}^{\text{SoP}}$[2]) and the SoP CS ($\mathbf{K}_{\text{CS}}^{\text{SoP}}$). We present the classification and clustering rates for both tasks.
2. Comparing their performance with their respective standard, non-randomized, versions: the LED ($\mathbf{K}_{\text{LED}}$) and the LCS ($\mathbf{K}_{\text{LCS}}$). Classification and clustering rates will be compared for all methods.
3. Comparing their performance with state-of-the-art kernels used for sequence similarity computation (Shawe-Taylor and Cristianini, 2004) such as the ASK ($\mathbf{K}_{\text{AS}}$) (Vishwanathan and Smola, 2003), the GASK ($\mathbf{K}_{\text{GAS}}$) (Lodhi et al., 2002), the FLK ($\mathbf{K}_{\text{FL}}$) (Watkins, 1999), the PSPECK ($\mathbf{K}_{\text{PSPEC}}$) (Leslie et al., 2002), as well as the SED ($\mathbf{K}_{\text{SED}}$) (Bahl and Jelinek, 1975; Oncina and Sebban, 2006) and a method for biological sequence comparison, KD ($\mathbf{K}_{\text{KD}}$) (Waterman and Eggert, 1987). Notice that only four of these methods do not depend on a parameter (see Table 2.3). Classification and clustering rates will be compared for all methods.

Please notice that the normalization and centering of the kernel matrices are also considered as meta-parameters (see Section 2.4.2 for more details). The local costs (or similarities) are set as follows: for edit distances $c_{ins} = c_{del} = c_{repl} = 1$ (for non-matching substitutions) and $c_{repl} = 0$ for matching substitutions; for common subsequences $s_{ins} = s_{del} = s_{subs} = 0$ (for non-matching substitutions) and $s_{subs} = 1$ for matching substitutions. Classification tasks on five real-world data sets have been performed as presented in the following sections. Notice also that in order to avoid overflow or underflow problems, we applied the standard formula for the logarithm of a sum (see, e.g., Huang et al. (1990)) in computing z_{1k} and z_{kn}.

[2]Note that the two proposed methods are not valid kernels but similarity measures. However, for simplicity, the letter **K** will be used indifferently throughout the document.

Algorithm and their abbreviations			**Parameter and tested values**
$\mathbf{K_{ED}^{SoP}}$	SoP ED	Sum-over-Paths edit distance	Inverse temperature $\theta = \{0.1, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4\}$
$\mathbf{K_{CS}^{SoP}}$	SoP CS	Sum-over-Paths common subsequence	Inverse temperature $\theta = \{0.1, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4\}$
$\mathbf{K_{SED}}$	SED	Stochastic edit distance	Error probability $p_e = \{0.05, 0.1, 0.15, 0.2, 0.25\}$
$\mathbf{K_{LED}}$	LED	Levenshtein edit distance	
$\mathbf{K_{LCS}}$	LCS	Longest common subsequence	
$\mathbf{K_{KD}}$	KD	K-best distinct alignments	
$\mathbf{K_{AS}}$	ASK	All Subsequences Kernel	
$\mathbf{K_{GAS}}$	GASK	Gap-weighted All Subsequences Kernel	Weight of gaps $\lambda = \{0.2, 0.4, 0.5, 0.6, 0.8\}$
$\mathbf{K_{FL}}$	FLK	Fixed Length Kernel	Length of substring $l = \{5, 10, 15, 20\}$
$\mathbf{K_{PSPEC}}$	PSPECK	p-Spectrum Kernel	Length of subsequence $p = \{5, 10, 15, 20\}$

Table 2.3: List of compared methods and parameter values tested in the internal cross-validation.

2.4.1 Data sets

This section introduces the different data sets used for the experiments and provide some examples (see Table 2.4) in order to give a more detailed insight of the performed experiments. Please note that all the described data sets are made available in `http://github.com/silviagdiez/thesis`. The data sets presented in this section have been chosen as they represent two of the most common applications of ASM: Optical Character Recognition (OCR), and comparison of biological sequences of proteins, in this case. The reason that we chose these particular subsets, was the availability of the data, either by collaboration within the same laboratory, or by publication on the Web.

Data set	**Data example**	**Example sequence representation**
Digits	1	224445455454555445466671001101011101011
Letters	b	8211111111111111111112269999999999963333332221111111447777778999
Figures	O	11111111111223333333333336699999999999999999988777777777771111111111
Arrows	←	141444411114477777777771111111111111111448
Proteins	Acetyltransferase	PAFYKKHGYKVIGVSEITPKGHNRYYLKKG

Table 2.4: Examples of samples of the tested data sets.

Digits data set. This data set was introduced by Oncina and Sebban (2006) and is originally based on the NIST Special Database 3 of the National Institute of Standards and Technology. A subset of 100 sequences of each digit (from 0 to 9), thus 1,000 sequences in total from 10 different writers, was extracted for our experiments. Each sequence was obtained by mapping its shape from the bitmap digit to a sequence of numbers that expresses the direction of the perimeter. An example of these data can be seen in Table 2.4.

Letters, figures, and arrows data sets. These three data sets were collected by Beuvens and T.Dullier (2009) from 30 different people who wrote ten times each letter from the English alphabet as well as a series of geometric shapes, such as arrows or other figures, on a digital tablet. Each of these three data sets is composed of sequences of numbers that represent the geographical directions of their perimeter, as described by Beuvens and T.Dullier (2009). An example of these data can be seen in Table 2.4. For the *letters data set*, we selected the sequences corresponding to the subset of the first 13 letters of the alphabet $\{a, b, c, d, e, f, g, h, i, j, k, l, m\}$. This is a balanced dataset with class priors of 7.7% each. The *arrows data set* corresponds to the set $\{\leftarrow, \uparrow, \rightarrow, \downarrow\}$ with a total number of sequences of 1,010 where the class distribution is $\{256, 252, 245, 257\}$. Sequences are extracted in the same way as for the letters data set. The last data set tested correspond to the *geometric figures* set. For this data set, 500 sequences comprising the following classes were collected: {*circle,triangle,rectangle,square,pentagon*}. This is a balanced set with class priors of 20% each. An example of these data can be seen in Table 2.4.

Proteins data set. This data set was collected from the UniProtKB and consists on five groups of similar proteins extracted via a BLAST query with default parameters on a *seed sequence*. The list of seed sequences as well as the proteins that integrate each of the groups are available on `http://github.com/silviagdiez/thesis`. Each class contains 50 similar sequences, and there are five classes in total, therefore providing a 250 balanced sequences data set.

2.4.2 Supervised classification methodology

Three types of supervised tasks have been performed over the whole data sets with the following classifiers:

1. A 1-NN based on random prototypes.
2. A 1-NN based on within-class centroids.
3. A SVM.

In each case, a 10-fold nested cross-validation has been applied. Estimated classification rates as well as their 95% confidence interval are reported.

The 1-NN based on random prototypes (1-NNP) assigns each observation to the class to which belongs his nearest prototype. A class prototype is a *randomly chosen observation* among all observations from a given class, repeating this procedure with 10 different prototypes, therefore reporting an average of the estimated classification rates.

In the case of the 1-NN based on within-class centroids (1-NNC), the observations are assigned to the class of its nearest centroid. A centroid is the *most central observation* among all observations from a given class.

In both cases, a grid cross-validation was performed in order to tune parameters (see Table 2.3) and kernel normalization choice, as explained below. This methodology also holds for the SVM[3], where the additional parameter C

[3]The chosen implementation was the LIBLINEAR package (Fan et al., 2008) with default parameters. The package can be downloaded from `http://www.csie.ntu.edu.tw/~cjlin/`

had to be tuned. The tested values were $C = \{0.1, 0.5, 1, 5, 10, 50, 100, 500, 1000\}$.

As explained, the normalization and/or centering of a kernel are considered as another parameter to be tuned during internal cross-validation (four kernels were computed for each value of the parameters, and the C value: no centering nor normalization, centering only, normalization only, and centering and then normalization of the kernel). In this way, normalized (and/or centered) kernels are only used when their performance proves better than the one of the original kernel. Please refer to Annex B for more details. These transformed kernels (centered and/or normalized) are referred to as normalized kernels for the sake of simplicity.

For this supervised classification task, simple nearest neighbor rules are used because the aim of the experiments, and more generally the work, is not to design a state-of-the-art sequence classifier; they rather aim at comparing different similarity measures between sequences. A good similarity measure should lead to compact, well-separated, classes in the embedding space. This is also the reason why we tested both the within-class centroids, and the random prototypes as class representative. A crude classifier, like the 1-NN we are using, will probably be more sensitive to the compactness/separability of the different classes in the embedding space while a more sophisticated classifier, like a SVM, could achieve good performance, even if the different classes are not well-separated and compact.

2.4.3 Results and test of hypothesis

Results obtained by nested cross-validation for all methods on all data sets are reported in Table 2.6. In order to verify that the proposed methods, SoP ED and SoP CS, perform better than their non-randomized versions, LED and LCS, a non-parametric one-sided paired sign test has been performed. The classification rates from the classification experiments for all data folds have been used as sample for this test (10-folds $\times$ 5 data sets = 50 samples

`liblinear`.

per classification method and algorithm). Results showed that, with a 95% confidence level (p-values), the proposed methods perform better in all cases, except in the case of the SVM for the SoP ED, where the confidence is 94%. Moreover, the SoP CS appears to perform slightly better that the SoP ED (with about 100% confidence level for the 1-NN and 84% for the SVM). The obtained p-values are shown in Table 2.5. While significant, the magnitude of the improvement is, however, not always spectacular on these investigated datasets (see Table 2.6).

Classification Method	**SoP ED vs. LED**	**SoP CS vs. LCS**	**SoP CS vs. SoP ED**
1-NN based on within-class centroids	0.0000	0.0147	0.0000
1-NN based on prototypes	0.0000	0.0000	0.0000
SVM	0.0551	0.0266	0.1537

Table 2.5: p-values for paired sign test for each classification method and algorithm.

2.4.4 Discussion of the results

The conclusions following this experimental study are rather clear; let us indeed return to our research questions. First, we can conclude that, in general, the SoP CS similarity measure and the SoP ED similarity measure based on the edit-distance usually perform significantly better than the original standard measures (standard LED and LCS), as shown by the test of hypothesis. It can be observed that when LCS performs better than LED, it is the randomized version, SoP CS, which will perform best (respectively for the LED and SoP ED).

Second, it has been observed that the SoP methods show an improvement over the remaining methods, be it on the OCR classification tasks or on the protein task.

Third, the string kernels tested in this work (the all subsequences kernel, the gap-weighted all subsequences kernel, the fixed length kernel, and the p-spectrum kernel) performed poorly in comparison with, for instance, a stan-

Classification results (1-NN using random class prototypes)

	Arrows Data Set	Digits Data Set	Figures Data Set	Letters Data Set	Proteins Data Set
$\mathbf{K}_{\mathbf{ED}}^{\mathbf{SoP}}$	73.59 ± 0.34%	66.37 ± 0.37%	35.10 ± 0.39%	**40.18 ± 0.22%**	**39.28 ± 0.53%**
$\mathbf{K}_{\mathbf{LED}}$	68.68 ± 0.88%	61.46 ± 0.36%	35.08 ± 0.22%	34.20 ± 0.21%	39.72 ± 0.83%
$\mathbf{K}_{\mathbf{SED}}$	68.76 ± 0.52%	63.35 ± 0.43%	34.38 ± 0.31%	36.05 ± 0.12%	37.88 ± 0.68%
$\mathbf{K}_{\mathbf{CS}}^{\mathbf{SoP}}$	81.26 ± 0.44%	72.59 ± 0.42%	37.28 ± 0.32%	45.26 ± 0.24%	37.28 ± 0.45%
$\mathbf{K}_{\mathbf{LCS}}$	**79.61 ± 0.71%**	**67.10 ± 0.34%**	**37.00 ± 0.29%**	39.61 ± 0.22%	34.96 ± 0.55%
$\mathbf{K}_{\mathbf{AS}}$	76.63 ± 0.83%	31.00 ± 0.36%	25.88 ± 0.47%	18.39 ± 0.63%	27.72 ± 0.47%
$\mathbf{K}_{\mathbf{FL}}$	72.79 ± 0.86%	32.34 ± 0.52%	28.58 ± 0.37%	22.79 ± 0.27%	26.60 ± 0.25%
$\mathbf{K}_{\mathbf{PSPEC}}$	68.32 ± 0.62%	33.57 ± 0.31%	34.86 ± 0.51%	16.54 ± 0.14%	36.56 ± 0.54%
$\mathbf{K}_{\mathbf{KD}}$	43.71 ± 0.42%	17.69 ± 0.35%	35.98 ± 0.64%	20.95 ± 0.33%	32.56 ± 0.59%
$\mathbf{K}_{\mathbf{GAS}}$	78.81 ± 0.53%	58.31 ± 0.30%	(*)	32.08 ± 0.33%	39.48 ± 0.54%

Classification results (1-NN using within-class centroids)

	Arrows Data Set	Digits Data Set	Figures Data Set	Letters Data Set	Proteins Data Set
$\mathbf{K}_{\mathbf{ED}}^{\mathbf{SoP}}$	96.34 ± 1.53%	91.60 ± 1.61%	56.80 ± 2.63%	70.96 ± 2.56%	71.20 ± 4.93%
$\mathbf{K}_{\mathbf{LED}}$	96.34 ± 1.64%	89.40 ± 1.90%	53.60 ± 1.92%	67.31 ± 2.69%	68.00 ± 4.53%
$\mathbf{K}_{\mathbf{SED}}$	96.04 ± 1.69%	90.60 ± 2.23%	53.00 ± 2.13%	68.94 ± 2.70%	66.00 ± 4.71%
$\mathbf{K}_{\mathbf{CS}}^{\mathbf{SoP}}$	**96.73 ± 1.78%**	95.00 ± 1.20%	63.60 ± 1.52%	77.69 ± 2.21%	**71.20 ± 5.07%**
$\mathbf{K}_{\mathbf{LCS}}$	97.13 ± 1.70%	**94.20 ± 1.20%**	**59.80 ± 2.22%**	**77.12 ± 1.45%**	70.00 ± 4.41%
$\mathbf{K}_{\mathbf{AS}}$	25.64 ± 0.79%	10.00 ± 0.00%	20.00 ± 0.00%	7.69 ± 0.00%	20.80 ± 1.05%
$\mathbf{K}_{\mathbf{FL}}$	74.26 ± 1.83%	46.40 ± 2.25%	29.60 ± 1.52%	20.58 ± 1.87%	28.80 ± 3.05%
$\mathbf{K}_{\mathbf{PSPEC}}$	80.59 ± 4.10%	54.70 ± 1.92%	43.00 ± 3.21%	24.52 ± 1.60%	58.40 ± 6.10%
$\mathbf{K}_{\mathbf{KD}}$	50.20 ± 2.15%	23.90 ± 2.05%	53.20 ± 2.81%	38.08 ± 2.52%	66.80 ± 5.49%
$\mathbf{K}_{\mathbf{GAS}}$	26.63 ± 0.94%	11.70 ± 0.59%	(*)	7.69 ± 0.00%	23.20 ± 2.28%

Classification results (SVM)

	Arrows Data Set	Digits Data Set	Figures Data Set	Letters Data Set	Proteins Data Set
$\mathbf{K}_{\mathbf{ED}}^{\mathbf{SoP}}$	98.51 ± 1.05%	97.90 ± 1.07%	**84.80 ± 1.46%**	**94.04 ± 0.78%**	73.20 ± 4.68%
$\mathbf{K}_{\mathbf{LED}}$	**98.61 ± 0.88%**	97.90 ± 1.03%	83.80 ± 1.23%	92.12 ± 1.75%	68.00 ± 6.61%
$\mathbf{K}_{\mathbf{SED}}$	98.32 ± 1.16%	97.40 ± 0.98%	87.80 ± 1.98%	94.42 ± 0.92%	**73.60 ± 4.85%**
$\mathbf{K}_{\mathbf{CS}}^{\mathbf{SoP}}$	98.71 ± 1.00%	**98.00 ± 1.05%**	83.00 ± 2.76%	93.56 ± 0.93%	74.80 ± 5.23%
$\mathbf{K}_{\mathbf{LCS}}$	98.22 ± 1.03%	98.30 ± 1.01%	83.80 ± 2.44%	91.83 ± 1.23%	70.40 ± 4.56%
$\mathbf{K}_{\mathbf{AS}}$	92.28 ± 1.07%	66.20 ± 4.83%	20.00 ± 0.00%	26.92 ± 4.68%	23.60 ± 4.44%
$\mathbf{K}_{\mathbf{FL}}$	96.24 ± 1.35%	80.30 ± 5.45%	76.20 ± 2.06%	72.31 ± 3.09%	36.00 ± 6.51%
$\mathbf{K}_{\mathbf{PSPEC}}$	97.52 ± 1.45%	77.80 ± 3.16%	59.80 ± 6.44%	59.33 ± 1.97%	68.40 ± 5.90%
$\mathbf{K}_{\mathbf{KD}}$	95.45 ± 1.20%	82.10 ± 1.86%	81.40 ± 3.51%	84.62 ± 2.37%	67.20 ± 5.82%
$\mathbf{K}_{\mathbf{GAS}}$	95.84 ± 1.19%	83.60 ± 8.54%	(*)	54.04 ± 3.83%	41.20 ± 4.68%

Table 2.6: Estimated classification rates with a 95% confidence interval obtained with the three classification methods. Best performing results are highlighted in grey, and second-best methods are marked in bold. The results marked with (*) have been omitted due to excessive computation time.

dard longest common subsequence measure. Here is a tentative explanation of this fact. As discussed in the related work from Section 2.2, we expect string kernels to perform well when the alphabet is very large (for instance in the context of text mining – in this case, there are very few long common subsequences), and worse for reduced alphabets.

It must also be noted the good performance of the stochastic edit distance (SED) in the case of the SVM for the letters and figures data sets. These two data sets contain the longest sequences among all tested data. This may indicate that the SED is a more performing measure for long sequence comparison combined with an SVM classifier. Yet another important property of the SED is that it is based on a log-likelihood instead of a expected cost (or similarity) for the SoP ED and CS (see Section 2.3.1 for more details). Finally, there is no clear winning string kernel for all the tested data sets.

In conclusion, these experiments show that the best method overall are the Sum-over-Paths distances. They consistently provide good results, they are almost always among the best method and, when is not the case, their performance is always very close to the winning method. Moreover, their computation complexity is similar to that of the standard edit distance and longest common subsequence algorithms, although at the price of a parameter, θ. Finally, the SoP common subsequence seems to provide slightly better results than the SoP edit distance.

Visualization of the effect of θ with a Kernel Principal-Component Analysis

Let us now turn to an example that visually shows the effect of θ on the separation between three subsets from the Digits data set (see Section 2.4.1 for more details on this data set). This experiment performs a Kernel Principal-Component Analysis (KPCA) on the similarity matrix obtained with the SoP CS with different θ values. The obtained projection is represented in a 3-dimensional space where each axis is one of the three main eigenvectors (or principal components) of the decomposition, and where each subset is repre-

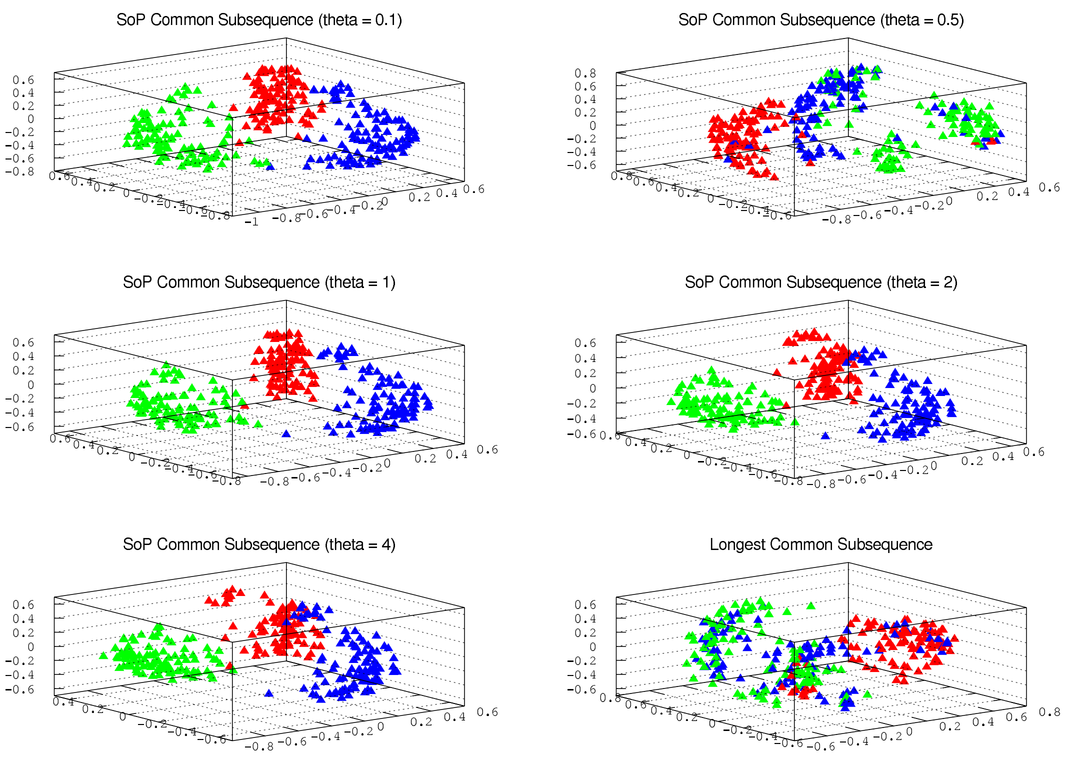


Figure 2.11: Kernel Principal-Component analysis based on the SoP CS with θ values of $\{0.1, 0.5, 1.0, 2.0, 4.0\}$ as well as the longest common subsequence. The three shown classes correspond to the handwritten digits: 0 (red), 1 (blue), and 4 (green) from the digits data set.

sented with a different color: the digit 0 observations are in red, the observations for digit 1 are in blue, and those for digit 4 are in green. We can observe from Figure 2.11 the evolution of the distribution of the three observation "clouds" as θ increases. Two conclusions can be extracted here: (i) the longest common subsequence (equivalent to the SoP CS with $\theta \to \infty$) does not provide the best separation between "clouds", and (ii) an intermediate value of θ gives a better separation, although this value depends on the data set.

2.5 The normalized Sum-over-Paths distances

As shown in the previous section, the SoP ED and the SoP CS are good performing measures in string classification and string clustering tasks. However, these measures suffer from normalization issues. A *normalized measure* is a measure that weights the similarity (or distance) of two strings with respect to their lengths. As stated by Marzal and Vidal (1993): "two (edit) errors in a comparison between strings of length 3, are more important than three errors in a comparison of strings of length 9" (see Table 2.7 for an example).

Strings	**Length**	**Distance**	**Normalized distance**
$\mathbf{s}_1$ = abcbadebc	9	$d(\mathbf{s}_1, \mathbf{s}_2) = 2$	$d_n(\mathbf{s}_1, \mathbf{s}_2) = 2/9 = 0.22$
$\mathbf{s}_2$ = eecbadebc	9		
$\mathbf{s}_3$ = abc	3	$d(\mathbf{s}_3, \mathbf{s}_4) = 2$	$d_n(\mathbf{s}_3, \mathbf{s}_4) = 2/3 = 0.66$
$\mathbf{s}_4$ = ddc	3		

Table 2.7: Example of normalized Levenshtein edit distance for comparison of two pairs of strings of different lengths. Differences between strings are marked in red, and shared sub-strings are marked in blue.

It is obvious from Table 2.7, that the distance d between $\mathbf{s}_1$ and $\mathbf{s}_2$ should be less important than the distance between $\mathbf{s}_3$ and $\mathbf{s}_4$. The normalized distance, d_n provides a more accurate measure of the difference between both strings. This principle also holds in the case of two strings with different lengths.

In order to overcome this problem, two straightforwardly normalized versions are introduced: the *Sum-over-Paths Normalized Edit Distance* (SoP NED), and the *Sum-over-Paths Normalized Common Subsequence* (SoP NCS).

2.5.1 Related work

Marzal and Vidal (1993) proposed a *Normalized Edit Distance* (NED) which represents the minimum ratio between the weight and the size of the alignment. They define γ as the edit function $\gamma(a \to b)$ that assigns to each editing operation transforming a into b a nonnegative real number. The NED is then defined as

$$\hat{W}(\wp) = \frac{W(\wp)}{L(\wp)} \tag{2.19}$$

where $W(\wp)$ is the weight of the path $\wp$ and $L(\wp)$ represents its length. They also show that, in order to $\hat{W}(\wp)$ be optimal, it is not sufficient to take the path with smallest $W(\wp)$ and divide it by its length (also called *post-normalization*); both parameters, $W(\wp)$ and $L(\wp)$ must be optimized at the same time. Furthermore, in order to keep the triangle inequality (see Table 2.2), post-normalization must be avoided and the choice of γ is restricted. An optimization of the computation of the NED is provided by Vidal et al. (1995) by applying fractional programming techniques which are an optimization technique for problems involving fractional or ratio functions.

Other computation methods are also proposed by Oommen and Zhang (1996) and Yujian and Bo (2007) by using related measures such as the *string-constrained* edit distance in the former, or the *generalized Levenshtein distance* (GLD) in the latter. However, the choice of γ may prevent the distance from being a metric or may increase the complexity of the algorithm. Furthermore, as suggested by Weigel and Fein (1994), the NED would favor longer alignments with cheaper operations than shorter ones with more expensive operations.

However, any of these measures takes into account the full space of possible editing paths between two sequences. For this reason, we will introduce in the next section the normalized version of the Sum-over-Paths distances and its computation.

2.5.2 Computation of the normalized Sum-over-Paths distances

Two ASM methods based on a post-normalization of the *Sum-over-Paths edit distance* (SoP ED) and the *Sum-over-Paths common subsequence* (SoP CS) are presented in this section. The main contribution of these new measures is to overcome the normalization issues that suffer both the SoP ED and the SoP CS without increasing the complexity of the algorithm. As a result we obtain two measures which:

1. Provide a model-independent measure of similarity taking all alignments into account.

2. Avoid noisy measures by favoring relevant subsequences of nearly-optimal alignments.

3. Overcome the normalization issues.

In order to normalize the SoP distances, a simple post-normalization of the expectation of the cost $\mathbb{E}[C]$ by the expectation of the length of the paths $\mathbb{E}[L]$ is applied. A fractional programming technique is of no use in our case, as this method focuses on a single path, over which it iterates until convergence to the path with minimum $\hat{W}(\wp)$. As the SoP distances rely on expectations of length and cost (using all possible paths in order to compute these quantities) it is not feasible to apply a pre-normalization which would consider the only path that minimizes this ratio. We therefore define the new SoP NED as follows

$$d_{\text{NSoP}} = \frac{\mathbb{E}[C]}{\mathbb{E}[L]} \tag{2.20}$$

where we recall from Equations (2.6) and (2.10) that

$$\mathbb{E}[C] = \sum_{k=1}^{n} \sum_{k'=1}^{n} c_{kk'} n_{kk'} \tag{2.21}$$

and we recall from Equation (1.26) (where $v_{kk'} = 1$ for all (k, k'))

$$\mathbb{E}[L] = \sum_{k=1}^{n} \sum_{k'=1}^{n} n_{kk'} \tag{2.22}$$

We therefore obtain

$$d_{\text{NSoP}} = \frac{\mathbb{E}[C]}{\mathbb{E}[L]} = \frac{\sum_{k=1}^{n} \sum_{k' \in \text{Succ}(k)} c_{kk'} n_{kk'}}{\sum_{k=1}^{n} \sum_{k' \in \text{Succ}(k)} n_{kk'}} \tag{2.23}$$

Now, by substituting with Equation (1.34), we obtain the final formula

$$d_{\mathrm{NSoP}} = \frac{\sum_{k=1}^{n} \sum_{k' \in \mathrm{Succ}(k)} z_{1k} c_{kk'} \exp[-\theta c_{kk'}] z_{k'n}}{\sum_{k=1}^{n} \sum_{k' \in \mathrm{Succ}(k)} z_{1k} \exp[-\theta c_{kk'}] z_{k'n}} \frac{z_{1n}}{z_{1n}} \tag{2.24}$$

$$= \frac{\sum_{k=1}^{n} \sum_{k' \in \mathrm{Succ}(k)} z_{1k} c_{kk'} \exp[-\theta c_{kk'}] z_{k'n}}{\sum_{k=1}^{n} \sum_{k' \in \mathrm{Succ}(k)} z_{1k} \exp[-\theta c_{kk'}] z_{k'n}} \tag{2.25}$$

where $c_{kk'} = 1$ for all (k, k') in $\mathbb{E}[L]$. We can similarly compute the SoP NCS as

$$s_{\mathrm{NSoP}} = \frac{\mathbb{E}[S]}{\mathbb{E}[L]} = \frac{\sum_{k=1}^{n} \sum_{k' \in \mathrm{Succ}(k)} s_{kk'} n_{kk'}}{\sum_{k=1}^{n} \sum_{k' \in \mathrm{Succ}(k)} n_{kk'}} \tag{2.26}$$

$$= \frac{\sum_{k=1}^{n} \sum_{k' \in \mathrm{Succ}(k)} z_{1k} s_{kk'} \exp[\theta s_{kk'}] z_{k'n}}{\sum_{k=1}^{n} \sum_{k' \in \mathrm{Succ}(k)} z_{1k} \exp[\theta s_{kk'}] z_{k'n}} \tag{2.27}$$

2.5.3 Experiments

This section aims at performing empirical experiments to:

1. Test the relative performance of the proposed methods.
2. Compare with the performance with their non-normalized versions.
3. Assess their goodness with respect to state-of-the-art kernels (Shawe-Taylor and Cristianini, 2004).

Several clustering and classification experiments on OCR data sets were performed. The *Letters*, *Arrows*, and *Digits* data sets are already introduced in Section 2.4.1. The fourth data set, *Images*, contains 417 horse images encoded by the k-nearest segment principle showing a horse (25%) or not (75%).

Compared methods include the LED, LCS, SoP ED, SoP CS, fixed-length subsequences kernel (FLK; see, e.g., Shawe-Taylor and Cristianini (2004)), p-spectrum kernel (PSPECK; see, e.g., Shawe-Taylor and Cristianini (2004)), all-subsequences kernel (ASK; see, e.g., Shawe-Taylor and Cristianini (2004)), gap-weighted all-subsequences kernel (GASK; see, e.g., Shawe-Taylor and Cristianini (2004)), as well as our two normalized methods: the SoP NED, and SoP NCS. Table 2.8 shows a summary of tested methods and parameters.

Clustering experiments with a kernel k-means were performed on three data sets. Parameter tuning is performed on an independent data set for those methods which need it (i.e., θ for the SoP methods). The clustering rate which measures the percentage of observations that were assigned to the right cluster or class after an optimal assignment is reported with 95% confidence. The rand index, and the obtained parameters are also reported in Table 2.9. Note that only the five best performing methods are shown in each case. Classification experiments with a SVM and a 1-NN were also performed. In this case, a nested cross-validation or an independent data set were used for tuning the parameters. Results are shown in Table 2.10, together with a 95% confidence interval. Please note that all kernel matrices were centered and normalized.

Algorithms and their abbreviations			**Parameter and tested values**
$\mathbf{K}_{\mathbf{ED}}^{\mathbf{SoP}}$	SoP ED	Sum-over-Paths edit distance	Inverse temperature $\theta = \{0.1, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4\}$
$\mathbf{K}_{\mathbf{CS}}^{\mathbf{SoP}}$	SoP CS	Sum-over-Paths common subsequence	Inverse temperature $\theta = \{0.1, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4\}$
$\mathbf{K}_{\mathbf{NED}}^{\mathbf{SoP}}$	SoP NED	Normalized Sum-over-Paths edit distance	Inverse temperature $\theta = \{0.1, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4\}$
$\mathbf{K}_{\mathbf{NCS}}^{\mathbf{SoP}}$	SoP NCS	Normalized Sum-over-Paths common subsequence	Inverse temperature $\theta = \{0.1, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4\}$
$\mathbf{K}_{\mathbf{LED}}$	LED	Levenshtein edit distance	
$\mathbf{K}_{\mathbf{LCS}}$	LCS	Longest common subsequence	
$\mathbf{K}_{\mathbf{AS}}$	ASK	All Subsequences Kernel	
$\mathbf{K}_{\mathbf{GAS}}$	GASK	Gap-weighted All Subsequences Kernel	Weight of gaps $\lambda = \{0.2, 0.4, 0.5, 0.6, 0.8\}$
$\mathbf{K}_{\mathbf{FL}}$	FLK	Fixed Length Kernel	Length of substring $l = \{5, 10, 15, 20\}$
$\mathbf{K}_{\mathbf{PSPEC}}$	PSPECK	p-Spectrum Kernel	Length of subsequence $p = \{5, 10, 15, 20\}$

Table 2.8: List of compared methods and parameter values tested.

Digits data set

	Classes = {2,8}	Classes = {3,7}	Classes = {2,5,7}	Classes = {6,7,8}	θ^*
	Clustering rates with 95% confidence interval				
$\mathbf{K}^{\mathbf{SoP}}_{\mathbf{NED}}$	100.00 ± 0.00%	100.00 ± 0.00%	99.33 ± 0.00%	98.67 ± 0.00%	2.0
$\mathbf{K}^{\mathbf{SoP}}_{\mathbf{ED}}$	99.50 ± 0.00%	98.00 ± 0.00%	99.00 ± 0.00%	98.67 ± 0.00%	2.0
$\mathbf{K}^{\mathbf{SoP}}_{\mathbf{NCS}}$	100.00 ± 0.00%	100.00 ± 0.00%	99.33 ± 0.00%	98.67 ± 0.00%	2.0
$\mathbf{K}^{\mathbf{SoP}}_{\mathbf{CS}}$	100.00 ± 0.00%	99.50 ± 0.00%	99.00 ± 0.00%	98.67 ± 0.00%	1.5
$\mathbf{K}_{\mathbf{LCS}}$	84.60 ± 0.13%	88.00 ± 0.00%	77.73 ± 0.09%	71.47 ± 0.14%	
	Adjusted rand index				
$\mathbf{K}^{\mathbf{SoP}}_{\mathbf{NED}}$	100.00 ± 0.00%	100.00 ± 0.00%	98.01 ± 0.00%	96.05 ± 0.00%	2.0
$\mathbf{K}^{\mathbf{SoP}}_{\mathbf{ED}}$	98.00 ± 0.00%	92.12 ± 0.00%	97.02 ± 0.00%	96.05 ± 0.00%	2.0
$\mathbf{K}^{\mathbf{SoP}}_{\mathbf{NCS}}$	100.00 ± 0.00%	100.00 ± 0.00%	98.01 ± 0.00%	96.05 ± 0.00%	2.0
$\mathbf{K}^{\mathbf{SoP}}_{\mathbf{CS}}$	100.00 ± 0.00%	98.00 ± 0.00%	97.02 ± 0.00%	96.05 ± 0.00%	1.5
$\mathbf{K}_{\mathbf{LCS}}$	47.63 ± 0.37%	57.56 ± 0.00%	44.46 ± 0.17%	49.85 ± 0.12%	

	Letters data set			**Arrows data set**		
	Classes = {b,m,o,z}		θ^*	Classes = $\{\leftarrow, \uparrow, \rightarrow, \downarrow\}$		θ^*
	Clustering rates and adjusted rand index with 95% confidence interval					
$\mathbf{K}^{\mathbf{SoP}}_{\mathbf{NED}}$	83.23 ± 0.00%	73.32 ± 0.00%	3	86.98 ± 0.46%	69.97 ± 0.87%	3.5
$\mathbf{K}^{\mathbf{SoP}}_{\mathbf{ED}}$	82.90 ± 0.00%	72.48 ± 0.00%	1	86.70 ± 0.53%	69.46 ± 1.01%	2.5
$\mathbf{K}^{\mathbf{SoP}}_{\mathbf{NCS}}$	83.55 ± 0.00%	74.17 ± 0.00%	2	88.86 ± 0.17%	73.66 ± 0.35%	3.5
$\mathbf{K}^{\mathbf{SoP}}_{\mathbf{CS}}$	83.55 ± 0.00%	74.17 ± 0.00%	4	89.16 ± 0.09%	74.25 ± 0.19%	3.5
$\mathbf{K}_{\mathbf{LCS}}$	81.88 ± 0.06%	64.15 ± 0.11%		89.51 ± 0.13%	75.10 ± 0.28%	

Table 2.9: Clustering results (clustering rates and adjusted rand index) obtained on three data sets for the five best-performing methods.

2.5.4 Discussion of the results

We can conclude from the results in Tables 2.9 and 2.10 that the SoP methods (SoP ED, SoP CS, SoP NED, SoP NCS) significantly outperform (1) their non-randomized version, i.e. LED and LCS, and (2) the tested string kernels, i.e. FLK, PSPECK, ASK, GASK. Moreover, the introduced normalized versions (SoP NED, SoP NCS) generally improve the existing results obtained by their non-randomized versions (SoP ED, SoP CS) while never being significantly worse.

Digits data set

	Classes = {2,8}	Classes = {3,7}	Classes = {2,5,7}	Classes = {6,7,8}	θ^*
	1-NN	1-NN	1-NN	1-NN	
	Classification rates with 95% confidence interval				
$\mathbf{K}^{\mathbf{SoP}}_{\mathbf{NED}}$	94.29 ± 0.87%	97.09 ± 0.56%	91.87 ± 1.12%	92.98 ± 0.80%	2.0
$\mathbf{K}^{\mathbf{SoP}}_{\mathbf{ED}}$	92.63 ± 0.89%	96.19 ± 0.53%	88.56 ± 1.52%	91.62 ± 0.90%	2.0
$\mathbf{K}^{\mathbf{SoP}}_{\mathbf{NCS}}$	95.24 ± 0.86%	97.59 ± 0.48%	92.71 ± 1.11%	93.65 ± 0.80%	2.0
$\mathbf{K}^{\mathbf{SoP}}_{\mathbf{CS}}$	94.00 ± 0.99%	97.13 ± 0.46%	91.83 ± 1.26%	93.10 ± 0.77%	1.5
$\mathbf{K}_{\mathbf{LCS}}$	87.13 ± 0.99%	91.47 ± 0.94%	77.67 ± 1.80%	81.19 ± 1.81%	

	Images data set	**Letters data set**		**Arrows data set**	
	Classes = {yes,no}	Classes = {b,m,o,z}		Classes = { $\leftarrow, \uparrow, \rightarrow, \downarrow$ }	
	SVM	SVM	1-NN	SVM	1-NN
	Classification rates with 95% confidence interval				
$\mathbf{K}^{\mathbf{SoP}}_{\mathbf{NED}}$	84.89 ± 0.02%	98.24 ± 0.01%	76.67 ± 0.07%	98.51 ± 0.01%	77.82 ± 0.05%
$\mathbf{K}^{\mathbf{SoP}}_{\mathbf{ED}}$	84.66 ± 0.03%	98.43 ± 0.01%	74.31 ± 0.08%	98.12 ± 0.01%	71.09 ± 0.06%
LED	83.71 ± 0.05%	–	–	–	–
$\mathbf{K}^{\mathbf{SoP}}_{\mathbf{NCS}}$	85.37 ± 0.03%	97.25 ± 0.02%	76.47 ± 0.07%	98.42 ± 0.01%	78.81 ± 0.04%
$\mathbf{K}^{\mathbf{SoP}}_{\mathbf{CS}}$	85.38 ± 0.03%	97.65 ± 0.02%	75.88 ± 0.06%	98.42 ± 0.01%	78.32 ± 0.04%
$\mathbf{K}_{\mathbf{LCS}}$	–	–	72.16 ± 0.07%	98.32 ± 0.01%	78.51 ± 0.07%
$\mathbf{K}_{\mathbf{FLK}}$	–	94.31 ± 0.02%	–	–	–

Table 2.10: Classification rates obtained on the four data sets for the five best-performing methods.

2.6 Conclusion

In this chapter we have introduced four Approximate String Matching methods based on the Sum-over-Paths formalism, the SoP ED, the SoP CS, and their normalized versions, the SoP NED and the SoP NCS. All these methods are based on a procedure for randomizing a dynamic programming algorithm defined on a lattice. It first defines a Boltzmann probability distribution on the set of paths through the lattice in such a way that good paths have a high probability of occurrence while bad paths have a low probability of being followed. Then, instead of computing the dynamic programming score on the optimal paths only, it averages the scores along all the possible paths, each individual score along a path being weighted by the probability of following it. This allows to account for the contributions of good, although sub-optimal, paths as well. Forward/backward recurrence relations allowing efficiently computation of the score are developed along the same line as the forward/backward algorithm in hidden Markov models.

Experimental results obtained on sequence classification tasks for the SoP ED and SoP CS indicate that, in some cases, taking the suboptimal alignments into account in the expected cost, improves significantly the results. In the remaining situations, it is the standard versions that perform best (the LED and the LCS). It must be noticed that in that case, its randomized version is the second winning method, and if we had extended the range for θ values, the performance would have been the same (as the standard measure is a specific case of the randomized one when $\theta \to \infty$).

This work also presented a normalized version of the SoP ED and SoP CS, the SoP NED and SoP NCS. The normalization of the SoP distances by the expected length of the alignments overcomes the issues due to the variation in length of the sequences being compared. Experimental results have shown the improvement in accuracy over the SoP ED and SoP CS techniques.

Future work will be devoted to the development of a valid positive semi-definite edit distance kernel based on the same ideas. We also plan to investigate the recently introduced free energy distance (Francoisse et al., 2013) in the context of sequence comparisons.

Chapter 3

A continuous-state version of discrete Randomized Shortest-Paths

Contents

Minimum-cost problems on a graph are of capital importance in a variety of problems, from robot path planning, to maze solving. Path planning (LaValle, 2006) is a well-known problem in the robotics community, described by (Steels, 1990) as "checking the consequences of an action in an internal model before performing such actions". However, it may be possible to have several initial and/or destination points.

In previous chapters, we have exploited the discrete RSP, where the state space is discrete and although there exists a notion of node ordering (some nodes can only be accessed from certain nodes), there is no notion of time. We present here the continuous-state counterpart of the discrete RSP, in which we assume a continuum of stages that fill densely the space, and where the notion of time is introduced.

The **research question** of this chapter is to investigate whether the continuous counterpart of the RSP framework can provide smooth, sub-optimal trajectories in path planning on multiple-source multiple-destination problems.

Path planning is the problem in which an agent needs to find a trajectory on a, possibly unkown space, respecting certain constraints such as optimality, avoidance of obstacles, smoothness of the trajectory, etc. Figure 3.1 presents an example of path planning where obstacles are avoided through (a) the shortest path, and (b) the safest path. The type of trajectory we usually want to achieve is the safest path, which allows the agent to move easily between obstacles and walls, while keeping a nearly optimal trajectory in terms of cost.

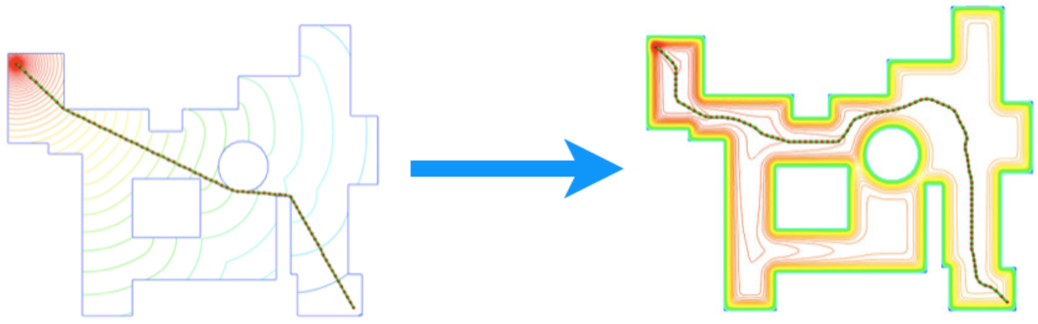


Figure 3.1: Path planning example: shortest path and safest path. Image extracted from `http://www.cvip.uofl.edu/wwwcvip/research/vision/mPathPlanning/`

We show the usefulness of the randomization framework of the RSP to achieve trajectories that are safe and sub-optimal at the same time. A Boltzmann probability distribution will be applied on the (usually infinite) set of paths connecting the source node(s) and the destination node(s), depending on an inverse temperature parameter θ. It is shown that the continuous-state counterpart requires the solution of two partial differential equations from which all the quantities of interest can be computed, e.g., the best local move is obtained by taking the gradient of the logarithm of one of these solutions. This will produce an optimal path, for a given θ, from source(s) to destination(s) avoiding obstacles. Examples are investigated for multiple-destination problems by computing the policy of an agent and verifying that the obtained trajectories met the initial requirements.

Main contributions

This chapter provides an optimal randomized policy based on the Sum-over-Paths framework for solving continuous-state path planning problems with multiple sources and multiple goals. It introduces a diffusion parameter for controlling the trade-off between exploration and exploitation, and it shows some interesting links between biased random walks on a graph (discrete RSP) and continuous-state Feynman-Kac diffusion processes.

3.1 Introduction

As seen in Chapter 1, the RSP framework defines a biased random walk on the graph that gradually favors low-cost paths as θ increases. This RSP approach is a discrete method that tackles the problem of finding the minimum-cost path on a graph while keeping a constant level of spread entropy (Saerens et al., 2009). The introduced path randomization allows balancing the load (number of packages) per path in the case of multiple goals, while exploiting those goals in parallel.

In order to compute the continuous-state counterpart of the RSP, we define a grid where each node has four neighbors (north, south, east, west) situated at a distance ϵ, and taking the limit $\epsilon \to 0$, a system of two independent partial differential equations computing forward and backward variables is obtained (Laplacian-based diffusion equations where the initial nodes are considered as sources and the destination nodes as sinks, and vice-versa). Once these variables are known, all the quantities of interest – such as the optimal randomized policy – can be easily computed. For instance, the best local move is obtained by taking the gradient of the logarithm of one of these solutions, namely the backward variable.

This chapter is organized as follows: in Section 3.2 we introduce the basic concepts of motion planning; Section 3.3 presents some of the related work; the continuous-state RSP extension is developed in Section 3.4; further details as well as as a physical interpretation and boundary conditions are here specified; the dynamic continuous-time continuous-state optimal policy is developed in Section 3.4.4; two practical, simulation examples, cases involving path planning are presented in Section 3.5; and Section 3.6 discusses the obtained conclusions and possible extensions.

3.2 Motion planning

Planning is a frequent task in the robotics domain, where a complex task needs to be decomposed into smaller moves that can be handled by a machine (Russell and Norvig, 2003). Let us consider, e.g., the problem of solving a maze where a starting point is fixed, and a path to the destination must be found through a series of obstacles. An example of this situation can is shown in Figure 3.2, where the shortest path between the initial and goal states (in yellow) is shown (in light blue) while avoiding the obstacles (in red). However, it is often desired to find a path that safely avoids these obstacles (see Figure 3.3 for an example) while complying with the optimality criterion, or that is able to handle multiple initial states and multiple source ones.

In the following sections we introduce the basic notions upon which is based the rest of this chapter, and the continuous-state version for the RSP is presented. This novel method provides us with a stochastic optimal policy for multiple-source multiple-destination problems.

3.2.1 Basic notions

Motion planning specifies how to move an object from an initial to a goal

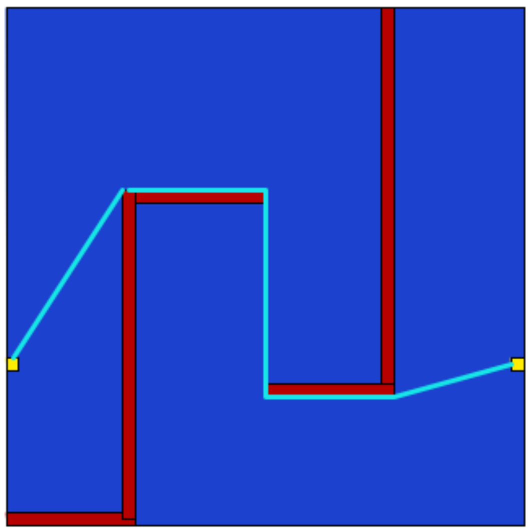


Figure 3.2: Shortest path in sample maze.

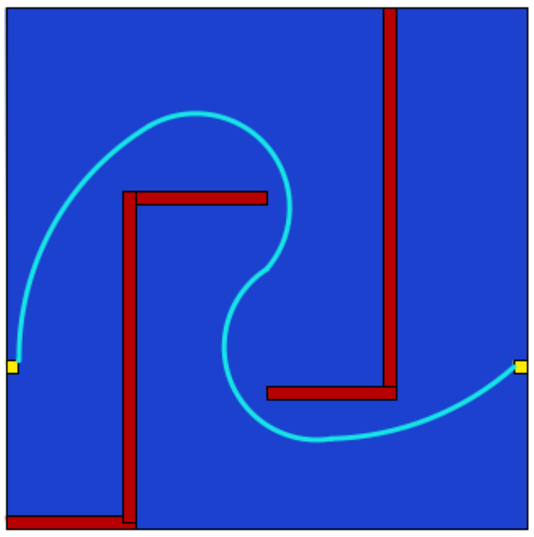


Figure 3.3: Safest path in sample maze.

state, while respecting the constraints of the object and the environment, e.g., the obstacles (Russell and Norvig, 2003). In *robotics*, motion planning focuses mainly on translations and rotations, taking into account the mechanical restrictions of the robot. In *control theory*, planning is achieved by designing the inputs to physical systems described by differential equations. In this domain, feedback policies and stability are of capital importance. Recently, the construction of inputs to non-linear dynamic systems have been used to bring an object from an initial to a goal state. On the other hand, planning in the *Artificial Intelligence* (AI) domain has a more discrete flavor.

Every produced plan, or *policy*, is then used by an *agent* that moves from an initial position, or *state*, to a goal state. Each state encodes the information of a position. The set of all possible states is called the *state space*, it is usually large, and cannot be completely explored. An agent takes *actions* according to this plan, and updates its state. These actions may be expressed in terms of a state-value function in a discrete space, or as differential equations in a continuous space or time. In planning, *time* can appear explicitly represented by fixing, e.g., a time constraint, although it is often expressed implicitly, e.g.,with the notion of sequential movements.

A plan must also comply with one of the following *criterion*:

- *Feasibility*: the plan brings the agent from the initial to the goal node with no optimization.
- *Optimality*: the plan must be feasible and optimal according to a certain criterion. In probabilistic uncertain domains, probabilities are often used as optimization criterion in terms of expected costs. This criterion applies to the approach presented in this chapter.

3.2.2 Planning in continuous spaces

Motion planning provides a plan that guides a robot from an initial to a goal state avoiding collisions in a continuous space. A problem that arises in this context is how to transform this continuous model into a discrete one. We can classify the different approaches among the following types (LaValle, 2006):

1. *Combinatorial motion planning*: this approach uses a discrete representation to exactly model the problem. However, this kind of method suffers from very large running times, as the state space is large, and its implementation is non-trivial.
2. *Sampling-based motion planning*: this approach uses collision-detection methods to sample the state space with further refinement through discrete searches. In this case, completeness[1] is not guaranteed.
3. *Application of numerical methods*: this method implicitly transforms a continuous space into a discrete one by applying, e.g, a finite difference approximation. By dividing the continuous space into a grid of homogeneous granularity, discrete actions can be produced for a certain state. This technique will be used for the continuous-state version of the RSP.

3.3 Related work

As presented in Sections 3.1 and 3.4.1, the continuous-state version of the RSP can be interpreted as a diffusion process. However, the use of physical analogies in path planning methods is not new. We can find three main types of physical analogies in the literature:

1. Wave propagation methods.
2. Potential field methods.
3. Diffusion strategies.

Wave propagation methods represent the first of the three main kinds of physical analogy in optimal path planning (Dautenhahn and Cruse, 1994; Rambidi and Yakovenchuck, 1999). Rambidi and Yakovenchuck (1999) introduce

[1] "An algorithm is *complete* if for any input it correctly reports whether there is a solution in a finite amount of time." (LaValle, 2006).

an analogue method for labyrinth solving based on parallel wave propagation through all possible paths in a reaction-diffusion media.

The second popular technique borrowed from physics are potential field methods (Connolly et al., 1990; Hwang and Ahuja, 1992; Khatib, 1986). Khatib (1986) proposes a real-time path planner based on an artificial potential field where the goal is represented as an attractive pole and the obstacles as repulsive faces. Similarly, the work from Connolly et al. (1990) proposes a smoothed version with two major advantages, i.e., it is based on a Laplace equation (which avoids local minima), and it benefits from the use of massively parallel architectures to solve this equation (efficient computation).

Eventually, diffusion strategies appeared as the third type of widely studied physical analogy in many path planning algorithms (Dautenhahn and Cruse, 1994; Li and Bui, 1998; Louste and Liegeois, 2000; Schmidt and Neubauer, 1992; Schmidt and Azarm, 1992; Steels, 1990; Tarassenko and Blake, 1991). A reaction-diffusion mechanism is presented by (Steels, 1990) in order to complete the behavior of a multi-agent model based on analogical representations. The propagation of an agent's information through his network of neighbors leads to the computation of a gradient field that will guide a robot on an obstacle grid. The DIP (Diffusion in Potential Fields) method, from Dautenhahn and Cruse (1994), computes a gradient field on a grid where each cell has an activation function which is computed by diffusion in a similar fashion to an automata's activation function.

The Laplace's equation is also used in diffusion strategies, e.g., the work by Schmidt and Neubauer (1992) and Schmidt and Azarm (1992) introduce the theoretical basis for a dynamic path planning approach using an unsteady diffusion equation with Dirichlet boundary conditions. This method enjoys the nice properties of Laplacian methods (high-speed, high efficiency), but also adapts to changing environments. A similar approach based on a fluid model represented by a Poisson equation with Newmann boundary conditions is presented by Li and Bui (1998).

Analogue systems have also adopted this strategy, as for instance the method from Tarassenko and Blake (1991), which represents obstacles as nonconducting solids in a conducting medium. More sophisticated methods, such as the one proposed by Louste and Liegeois (2000), cope with uneven natural terrain path planning. In this case, a viscous fluid formalism where external forces and friction are taken into account is used for multiple-source multiple-destination problems.

Kappen et al. (2012) introduced a class of stochastic optimal control problems in which the control is expressed as a probability function over future trajectories, where the control cost can be expressed as a Kullback-Leibler divergence and some interaction terms. In their work, they show how this KL control theory contains the path integral control method as a special case.

The continuous-state RSP version presented here belongs to the diffusion methods involving Laplacian differential operators with Dirichlet boundary conditions for multiple-source multiple-destination problems. Its main drawback is that paths are considered as independent and its most interesting properties are the fact that:

- it depends on a diffusion parameter controlling the **trade-off between exploration and exploitation**,
- the resulting policy is **optimal** since it ensures minimal expected cost for a constant exploration,
- it provides the minimum-cost policy when the **diffusion parameter** is low, and
- it shows some interesting links between biased random walks on a graph (discrete RSP) and continuous-state **Feynman-Kac diffusion processes**.

3.4 The continuous state-space equivalent to RSP

In this section we show how to adapt the discrete RSP framework from Section 1.1 to the continuous-state domain. In order to answer this question, let us consider a two-dimensional undirected lattice, with $c_{kk'} = c_{k'k}$, on which we apply the RSP framework. Each node has four neighbors as displayed in Figure (3.4) at a distance ϵ from each other.

The idea is to let the grid become dense by taking the limit $\epsilon \to 0$. The first step is to study the behavior of the forward/backward variables when taking this limit.

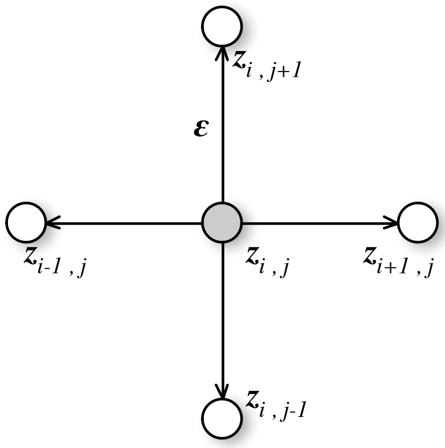


Figure 3.4: Forward/backward variable in a grid configuration with 4 neighbors. The arrows do not imply direction (this is an undirected lattice).

Let us recall that forward/backward variables are provided by Equations (1.18–1.19). Choosing uniform reference probabilities, $p^{(0)}_{kk'} = 1/4$ for all k, the new forward variable is obtained as follows

$$\begin{cases} z^f_1 = 1 + \displaystyle\sum_{k \in N(1)} \frac{1}{4} \exp[-\theta c_{k1}]\, z^f_k \\ z^f_{k'} = \displaystyle\sum_{k \in N(k')} \frac{1}{4} \exp[-\theta c_{kk'}]\, z^f_k, \text{ for } k' \neq 1 \end{cases} \tag{3.1}$$

and the new backward variable reads

$$\begin{cases} z_n^b = 1 + \displaystyle\sum_{k' \in N(n)} \frac{1}{4} \exp[-\theta c_{nk'}]\, z_{k'}^b \\ z_k^b = \displaystyle\sum_{k' \in N(k)} \frac{1}{4} \exp[-\theta c_{kk'}]\, z_{k'}^b, \text{ for } k \neq n \end{cases} \tag{3.2}$$

where $N(k)$ is the set of neighbors of node k, i.e., if $k = z_{i,j}$, then $N(z_{i,j}) = \{z_{i+1,j}, z_{i,j+1}, z_{i-1,j}, z_{i,j-1}\}$.

3.4.1 Computation of the forward/backward variables

As we have seen in the previous section, these forward/backward variables from which we can extract the quantities of interest from Chapter 1, depend on the local costs, $c_{kk'}$, and on themselves. In this section we will present the method for computing the new, continuous-state, variables.

We first consider the *forward equation*, Equation (3.1), and assume that each node on the grid is separated from its neighbors by a distance $\epsilon > 0$ (see Figure (3.4)). The forward variable z_k^f will be indexed by its position (x_k, y_k) and written as $z^f(x_k, y_k)$. In that case, the total cost along the path $\mathbf{r}(s) = (x(s), y(s))$ connecting node k to node k' is

$$c_{kk'} = \int_{(x_k, y_k)}^{(x_{k'}, y_{k'})} V(x(s), y(s)) ds \tag{3.3}$$

where $V(x, y) \geq 0$ is the *cost density* at (x, y) and s is the total displacement *along the trajectory* (its length)[2]. In other words, it is assumed that the cost is only related to the position of the walker and not his direction. We will, therefore, consider for simplicity that the cost $c_{kk'}$ is no more associated to the transition $k \to k'$, but only to the state k, $c_{kk'} = c_k$. Taking directions into account would require the use of tensors, which is not investigated in the present work.

[2]Note that, for the sake of readability, we also denote $V(x, y)$, as well as the other variables, as $V_{x,y}$ or $V(\mathbf{r})$.

As for any continuous-state stochastic process (Berg, 1993; Chaichian and Demichev, 2001; Gardiner, 2002; Holmes, 2002; Jacobs, 2010; Rudnick and Gaspari, 2004), let us now assume that $\epsilon \to 0$ while maintaining the ratio

$$\epsilon^2/\delta s = c \tag{3.4}$$

constant and finite, which means that in order to achieve a *net displacement* of ϵ, the random walker needs to make a total travel length of the order $\delta s \propto \epsilon^2$.

This implies that the total length of the path followed by the random walker is of considerably larger magnitude than the final net displacement, ϵ (Berg, 1993). When $\epsilon \to 0$,

$$c_{kk'} \simeq V(x_k, y_k)\delta s \tag{3.5}$$

and Equation (3.1) can be rewritten for the grid of Figure (3.4) as

$$\begin{aligned} z^f_{x,y} &= \frac{\exp[-\theta V_{x,y}\delta s]}{4}\, z^f_{x+\epsilon,y} + \frac{\exp[-\theta V_{x,y}\delta s]}{4}\, z^f_{x-\epsilon,y} \\ &\quad + \frac{\exp[-\theta V_{x,y}\delta s]}{4}\, z^f_{x,y+\epsilon} + \frac{\exp[-\theta V_{x,y}\delta s]}{4}\, z^f_{x,y-\epsilon} \\ &= \frac{\exp[-\theta V_{x,y}\delta s]}{4}[z^f_{x+\epsilon,y} + z^f_{x-\epsilon,y} + z^f_{x,y+\epsilon} + z^f_{x,y-\epsilon}] \end{aligned} \tag{3.6}$$

Expanding each term up to the second order of ϵ, e.g.,

$$z^f_{x-\epsilon,y} = z^f_{x,y} - \frac{\partial z^f_{x,y}}{\partial y}\epsilon + \frac{1}{2}\frac{\partial^2 z^f_{x,y}}{\partial y^2}\epsilon^2 + o(\epsilon^3) \tag{3.7}$$

provides

$$z^f_{x,y} = \frac{\exp[-\theta V_{x,y}\delta s]}{4}\left(4\, z^f_{x,y} + \frac{\partial^2 z^f_{x,y}}{\partial x^2}\epsilon^2 + \frac{\partial^2 z^f_{x,y}}{\partial y^2}\epsilon^2 + o(\epsilon^3)\right) \tag{3.8}$$

Keeping in mind that $\delta s = \epsilon^2/c$ and further expanding

$$\exp[-\theta V_{x,y}\delta s] = (1 - \frac{\theta}{c}V_{x,y}\epsilon^2) + o(\epsilon^3), \tag{3.9}$$

we obtain

$$z^f_{x,y} = \frac{1}{4}(1 - \frac{\theta}{c}V_{x,y}\epsilon^2 + o(\epsilon^3)) \times \left(4\, z^f_{x,y} + \frac{\partial^2 z^f_{x,y}}{\partial x^2}\epsilon^2 + \frac{\partial^2 z^f_{x,y}}{\partial y^2}\epsilon^2 + o(\epsilon^3)\right) \tag{3.10}$$

Without loss of generality, the constant c can be absorbed by θ: we now choose the units of θ in such a way that $c = 1$. Then, by defining the *diffusion constant* as $D = 1/(4\theta)$ and keeping only the terms in ϵ^2,

$$\frac{\partial^2 z^f_{x,y}}{\partial x^2} + \frac{\partial^2 z^f_{x,y}}{\partial y^2} = \frac{1}{D}V_{x,y}z^f_{x,y} \tag{3.11}$$

or

$$D\Delta z^f_{x,y} = V_{x,y}\, z^f_{x,y} \tag{3.12}$$

This is exactly the stationary solution of a Schrodinger-like diffusion equation without the imaginary term:

$$\mu\frac{\partial z^f(\mathbf{r}, t)}{\partial t} = D\Delta z^f(\mathbf{r}, t) - V(\mathbf{r})z^f(\mathbf{r}, t) \tag{3.13}$$

where $V(\mathbf{r})$ plays the role of a potential and $\mathbf{r}(\tau) = (x(\tau), y(\tau))$. Equation (3.13) is also sometimes called the Bloch equation (Chaichian and Demichev, 2001) in physics. It corresponds to a simple diffusion process for which the particle can disappear with a probability density $V(\mathbf{r})$ per unit of time at position $\mathbf{r}$, up to a normalization factor.

A diffusion process interpretation

There exists, indeed, an intuition related to general diffusion processes behind this equation (Berg, 1993; Chaichian and Demichev, 2001; Holmes, 2002). The well-known first Fick's law states that particle flow, $\mathbf{j}$, of a diffusing material in any part of the system is proportional to the local density of particle gradient (see, e.g., Berg (1993); Chaichian and Demichev (2001); Holmes (2002)). In other words,

$$\mathbf{j}(\mathbf{r}, t) = -D\boldsymbol{\nabla}\rho_t(\mathbf{r}) \tag{3.14}$$

where $\rho_t(\mathbf{r})$ is the particle density at time t and position $\mathbf{r} = (x, y)$, D is the diffusion constant, and $\mathbf{j}$ denotes the particle flow, i.e., $\mathbf{j} \cdot \mathbf{n}$, with $\|\mathbf{n}\| = 1$, is the net number of diffusing particles per unit of time passing through position $\mathbf{r}$ in the direction of $\mathbf{n}$. This principle is illustrated in Figure 3.5, where the flow of particles, $\mathbf{j}$, moves from the zone of highest particle concentration to the area with lowest one (the gradient of particle density).

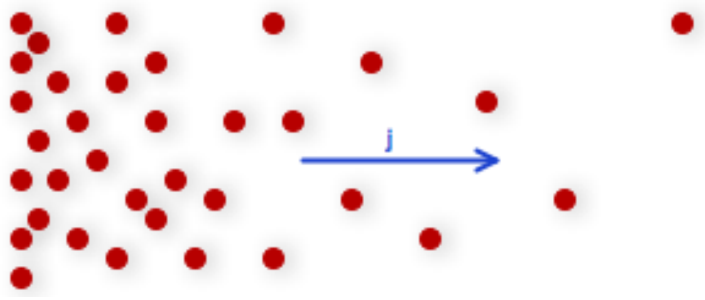


Figure 3.5: Example of diffusion of particles where $\mathbf{j}$ denotes the particle flow.

Furthermore, if particles are neither created nor destroyed, the basic continuity relations (Chaichian and Demichev, 2001; Holmes, 2002; Rudnick and Gaspari, 2004) in two dimensions are verified (see Arfken and Weber (2005) for standard notations)

$$\frac{\partial}{\partial t} \iint_{\Omega} \rho_t(\mathbf{r})\, dxdy = -\oint_{\partial\Omega} \mathbf{j}(\mathbf{r}, t) \cdot d\boldsymbol{\sigma} \tag{3.15}$$

where $\partial\Omega$ is the region boundary and $d\boldsymbol{\sigma}$ is the infinitesimal contour vector directed to the outside of $\partial\Omega$. Or, equivalently, from the divergence theorem,

$$\frac{\partial\rho_t(\mathbf{r})}{\partial t} = -\text{div}\, \mathbf{j}(\mathbf{r}, t) \tag{3.16}$$

Combining Fick's law with the continuity Equation (3.16) yields

$$\frac{\partial\rho_t(\mathbf{r})}{\partial t} = D\Delta\rho_t(\mathbf{r}) \tag{3.17}$$

Assume now that, instead of Equation (3.16), the density of particles is governed by

$$\frac{\partial \rho_t(\mathbf{r})}{\partial t} = -\mathrm{div}\, \mathbf{j}(\mathbf{r}, t) - V(\mathbf{r})\rho_t(\mathbf{r}) \tag{3.18}$$

which considers that particles are disappearing with a density $V(\mathbf{r})$ per unit of time (Chaichian and Demichev, 2001). This mimics the "evaporating" random walk behavior of the discrete RSP described in Section 1.5. The resulting equation is

$$\frac{\partial \rho_t(\mathbf{r})}{\partial t} = D\Delta\rho_t(\mathbf{r}) - V(\mathbf{r})\rho_t(\mathbf{r}) \tag{3.19}$$

which is exactly Equation (3.13). In addition, when an external force $\mathbf{f}$ is present – implying a drift (velocity) in the direction of $\mathbf{f}$ – this results in an additional flow of the form

$$\mathbf{j}_f(\mathbf{r}, t) = -\gamma\rho_t(\mathbf{r})\,\mathbf{f} \tag{3.20}$$

with γ being a mobility coefficient, the inverse of the friction coefficient (Berg, 1993; Chaichian and Demichev, 2001). The flow $\mathbf{j}$ of Equation (3.14) therefore becomes $\mathbf{j}(\mathbf{r}, t) = -D\boldsymbol{\nabla}\rho_t(\mathbf{r}) - \gamma\rho_t(\mathbf{r})\,\mathbf{f}$, yielding

$$\frac{\partial \rho_t(\mathbf{r})}{\partial t} = D\Delta\rho_t(\mathbf{r}) - \gamma\,\mathrm{div}(\rho_t(\mathbf{r})\,\mathbf{f}) - V(\mathbf{r})\rho_t(\mathbf{r}) \tag{3.21}$$

instead of (3.19). This equation will be encountered later, when the optimal policy is derived (see Equation (3.50)). Interestingly, it can be shown that the solution to Equation (3.19) is provided by (Chaichian and Demichev, 2001; Del Moral, 2004; Mazo, 2002)

$$\mathbb{E}_W\left[\exp[-\int_0^t V(x(\tau), y(\tau))d\tau]\right] \tag{3.22}$$

where $\mathbb{E}_W$ represents the expectation according to the Wiener measure. This corresponds to the celebrated Feynman-Kac formula which states that the solution to (3.19) can be interpreted as the expectation on all possible paths, each path being weighted by the exponential of minus the total cost cumulated along the path. Therefore, low-cost paths are favored with respect to high-cost paths. The stochasticity of the process (the exploration) is regulated by the diffusion constant D. The discrete RSP can therefore be considered as a discrete-state discrete-time counterpart of the Feynman-Kac diffusion process as well as the Bloch equation.

Initial and boundary conditions

We still have to precise the initial conditions and the boundary conditions. By looking at Equation (3.1), it can be seen that there is a unit source at node 1. Denoting the position of this source node 1 as (x_f, y_f) (the subscript f is for forward), the source becomes a delta of Dirac centered at this location. The coefficient multiplying this delta of Dirac is computed in the Appendix, and is equal to $-4D$. The forward stationary equation (3.12) becomes

$$\Delta z^f(x,y) = \frac{1}{D} V(x,y) z^f(x,y) - 4\delta(x - x_f)\delta(y - y_f) \tag{3.23}$$

with $D = 1/(4\theta)$. It can be observed that D plays the same role as a temperature (inverse of θ).

Exactly the same reasoning applies to the *backward variable*, and the partial differential equation easily follows

$$\Delta z^b(x,y) = \frac{1}{D} V(x,y) z^b(x,y) - 4\delta(x - x_b)\delta(y - y_b) \tag{3.24}$$

Concerning the boundary conditions, a barrier is produced by defining an infinite cost on the boundaries, preventing the random walkers from reaching them. This allows to mimic the discrete situation of the RSP on a graph (see Saerens et al. (2009) or Section 1.1). Dirichlet boundary conditions stating that both z^f and z^b are equal to zero on the boundary are therefore used. Thus, both in the continuous and the discrete case, an internal boundary layer $\partial\Omega$ is added with $V(x,y) = \infty$ for $(x,y) \in \partial\Omega$ in the continuous case, and $c_{kk'} = \infty$ for $k' \in \partial\Omega$ for the discrete one.

Notice that if we want to solve Equations (3.23-3.24) numerically by using a simple finite difference approximation with a central difference method, we exactly obtain Equations (1.18–1.19) with $\theta = f(D)$, some function of the diffusion constant. Let us for instance examine Equation (3.24). At the interior of the domain, $\Omega \backslash \partial\Omega$, the difference equation corresponding to (3.24) is

$$D(z^b_{x+\epsilon,y} + z^b_{x-\epsilon,y} + z^b_{x,y+\epsilon} + z^b_{x,y-\epsilon} - 4z^b_{x,y}) = \epsilon^2 V_{x,y} z^b_{x,y} \tag{3.25}$$

Isolating $z^b_{x,y}$ in this last equation provides

$$\left(4 + \frac{\epsilon^2 V_{x,y}}{D}\right) z^b_{x,y} = z^b_{x+\epsilon,y} + z^b_{x-\epsilon,y} + z^b_{x,y+\epsilon} + z^b_{x,y-\epsilon} \tag{3.26}$$

Assuming a small ϵ^2 in comparison with the value of D, this last equation is similar (up to the order $o(\epsilon^4)$) to

$$4 \exp\left[\frac{\epsilon^2 V_{x,y}}{4D}\right] z^b_{x,y} = z^b_{x+\epsilon,y} + z^b_{x-\epsilon,y} + z^b_{x,y+\epsilon} + z^b_{x,y-\epsilon} \tag{3.27}$$

which, in turn, is equivalent to the discrete counterpart (1.19) as shown now. Indeed, considering a sufficient dense grid with a small ϵ, $c_{kk'} \simeq V(x_k, y_k)\delta s$, let us rewrite Equation (1.19) as

$$4 \exp[\theta V(x_k, y_k)\, \delta s]\, z^b_k = \sum_{k' \in N(k)} z^b_{k'} \tag{3.28}$$

Evaluating the z^b_k on the grid yields

$$4 \exp[\theta V_{x,y}\, \delta s]\, z^b_{x,y} = z^b_{x+\epsilon,y} + z^b_{x-\epsilon,y} + z^b_{x,y+\epsilon} + z^b_{x,y-\epsilon} \tag{3.29}$$

which, by using $\delta s = \epsilon^2$, corresponds exactly to Equations (3.27, 1.19).

Therefore, the RSP framework can be considered as the discrete-state counterpart of the continuous-state Feynman-Kac process. The expected number of visits at any position of the grid, as well as the expected cost, are derived in the next subsection.

3.4.2 Computation of some of the basic quantities

We are ready now to compute the quantities of interest for the continuous-state framework. The continuous-state equivalent of the discrete *partition function* $\mathcal{Z} = z_{1n}$ is $z^f(x_b, y_b) = z^b(x_f, y_f)$. From Equation (1.15), the *expected number of visits* to position (x, y) when following the optimal policy is

$$n(x,y) = \frac{z^f(x,y)\, z^b(x,y)}{z^f(x_b,y_b)}, \text{ for } (x,y) \neq (x_f, y_f) \tag{3.30}$$

The *expected cost*, initially computed with Equation (1.21), for reaching (x_b, y_b) from (x_f, y_f) is provided by

$$\overline{C} = \frac{\iint_\Omega z^f(x,y) z^b(x,y)\, V(x,y)\, dx\, dy}{z^f(x_b,y_b)} \tag{3.31}$$

3.4.3 Derivation of the source term coefficient

Let us now derive the source term coefficient of Equations (3.23-3.24). By rewriting the discrete equation for the forward variable on the grid (Equation (3.6)) including the source term, we obtain

$$z^f_{x,y} = \frac{\exp[-\theta V_{x,y} \delta s]}{4} [z^f_{x+\epsilon,y} + z^f_{x-\epsilon,y} + z^f_{x,y+\epsilon} + z^f_{x,y-\epsilon}] + \delta(x,x_f)\delta(y,y_f) \tag{3.32}$$

where $\delta(x, x_f), \delta(y, y_f)$ are Kronecker deltas. Expanding the different terms up to the second order as in Section 3.4.1, Equation (3.10), yields

$$z^f_{x,y} = \frac{(1 - \theta V_{x,y}\epsilon^2 + o(\epsilon^3))}{4} \left[4\, z^f_{x,y} + \Delta z^f_{x,y}\epsilon^2 + o(\epsilon^3)\right] + \delta(x,x_f)\delta(y,y_f) \tag{3.33}$$

Thus, by rearranging this last equation, we obtain

$$\delta(x,x_f)\delta(y,y_f) = (\theta V_{x,y} z^f_{x,y} - \frac{1}{4}\Delta z^f_{x,y})\epsilon^2 + o(\epsilon^3) \tag{3.34}$$

Now, by summing this last equation over the entire grid and taking the limit $\epsilon \to 0$ with $dx = dy = \epsilon$ provides

$$1 = \sum_{x,y} \delta(x, x_f)\delta(y, y_f) \tag{3.35}$$

$$= \sum_{x,y} [(\theta V_{x,y} z^f_{x,y} - \frac{1}{4}\Delta z^f_{x,y})\epsilon^2 + o(\epsilon^3)] \tag{3.36}$$

$$\simeq \frac{1}{\epsilon^2} \int_{x,y} dxdy\, [(\theta V_{x,y} z^f_{x,y} - \frac{1}{4}\Delta z^f_{x,y})\epsilon^2 + o(\epsilon^3)] \tag{3.37}$$

$$= \int_{x,y} dxdy\, (\theta V_{x,y} z^f_{x,y} - \frac{1}{4}\Delta z^f_{x,y}) + o(\epsilon) \tag{3.38}$$

Therefore,

$$\theta V_{x,y} z^f_{x,y} - \frac{1}{4}\Delta z^f_{x,y} = \delta(x - x_f)\delta(y - y_f) \tag{3.39}$$

with $\delta(x - x_f), \delta(y - y_f)$ being Dirac deltas, and since $D = 1/(4\theta)$, we finally obtain Equation (3.23)

$$D\Delta z^f_{x,y} = V_{x,y} z^f_{x,y} - 4D\delta(x - x_f)\delta(y - y_f) \tag{3.40}$$

3.4.4 The dynamic, global, optimal, policy

In the previous sections we have introduced the motivation for planning algorithms, and how physical analogies can be applied to solve this problem. Furthermore, we have explained how to adapt the RSP discrete framework to the continuous-state version, and how to compute some of the quantities of interest. In this section we explain how to compute the policy or plan, which allows the agent to know which is the optimal action to take in a given state. Please recall that this is a stochastic policy, so it is optimal in terms of expected cost.

For deriving the optimal policy, let us consider the continuous-time, continuous-state, dynamics of a random walker following the optimal policy provided by Equation (1.16), or Equation (1.17) for the one-step ahead policy, in the discrete case. It is assumed that $z^b(x, y)$ is known (computed through Equation

(3.24)) and that the time is provided by the total displacement along the trajectory, i.e. $\delta t = \delta s$ (a unit velocity). The objective is to derive the probability density, $\rho^*_t(x, y)$, of finding the random walker in position (x, y) at time t when starting from some position $\rho^*_0(x, y) = \delta(x - x_0)\delta(y - y_0)$ and following the optimal randomized policy given by Equation (1.16), i.e., the continuous-state continuous-time counterpart of Equation (1.17).

It takes to the random walker a time $\delta t = \delta s = \epsilon^2$ to make a net displacement of ϵ so that taking a time step of δt and evaluating Equation (1.17) at $t + \delta t$ on position (x, y) of the two-dimensional grid (see Figure (3.4)), as well as assuming that (x, y) is in the interior of Ω so that the term δ_{kn} in the denominator of Equation (1.17) cancels, yields

$$\begin{aligned}
\rho^*_{t+\delta t}(x, y) = {} & \frac{\exp[-\theta V_{x+\epsilon,y}\delta s]}{4} \frac{z^b_{x,y}}{z^b_{x+\epsilon,y}} \rho^*_t(x + \epsilon, y) \\
& + \frac{\exp[-\theta V_{x-\epsilon,y}\delta s]}{4} \frac{z^b_{x,y}}{z^b_{x-\epsilon,y}} \rho^*_t(x - \epsilon, y) \\
& + \frac{\exp[-\theta V_{x,y+\epsilon}\delta s]}{4} \frac{z^b_{x,y}}{z^b_{x,y+\epsilon}} \rho^*_t(x, y + \epsilon) \\
& + \frac{\exp[-\theta V_{x,y-\epsilon}\delta s]}{4} \frac{z^b_{x,y}}{z^b_{x,y-\epsilon}} \rho^*_t(x, y - \epsilon)
\end{aligned} \tag{3.41}$$

Remembering that $\delta s = \delta t = \epsilon^2$, and expanding $\rho^*_{t+\delta t}(x, y)$ as well as the exponentials up to ϵ^2 gives

$$\begin{aligned}
\rho^*_t(x, y) + \frac{\partial \rho^*_t(x, y)}{\partial t}\epsilon^2 + o(\epsilon^3) = {} & \frac{z^b_{x,y}\,(1 - \theta V_{x,y}\epsilon^2 + o(\epsilon^3))}{4} \\
\times & \underbrace{\left[\underbrace{\frac{\rho^*_t(x + \epsilon, y)}{z^b_{x+\epsilon,y}}}_{(i)} + \underbrace{\frac{\rho^*_t(x - \epsilon, y)}{z^b_{x-\epsilon,y}}}_{(ii)} + \underbrace{\frac{\rho^*_t(x, y + \epsilon)}{z^b_{x,y+\epsilon}}}_{(iii)} + \underbrace{\frac{\rho^*_t(x, y - \epsilon)}{z^b_{x,y-\epsilon}}}_{(iv)} \right]}_{[(i)+(ii)+(iii)+(iv)]}
\end{aligned} \tag{3.42}$$

We now develop the terms (i)-(iv) appearing in the second line of the previous equation (3.42). For instance, the first term (i) is

$$\frac{\rho_t^*(x+\epsilon,y)}{z^b_{x+\epsilon,y}} = \frac{\rho_t^*(x,y)}{z^b_{x,y}} + \frac{\partial}{\partial x}\left(\frac{\rho_t^*(x,y)}{z^b_{x,y}}\right)\epsilon + \frac{1}{2}\frac{\partial^2}{\partial x^2}\left(\frac{\rho_t^*(x,y)}{z^b_{x,y}}\right)\epsilon^2 + o(\epsilon^3) \tag{3.43}$$

We immediately observe that terms (i)-(iv) of order ϵ in Equation (3.42) cancel out because they are evaluated both at $+\epsilon$ and $-\epsilon$. Therefore, dropping the dependency on (x, y) and referring $\partial z(x, y)/\partial x$ as $\partial_x z$ for convenience,

$$[(i) + ... + (iv)] = 4\frac{\rho_t^*}{z^b} + \left[\partial_x^2\left(\frac{\rho_t^*}{z^b}\right) + \partial_y^2\left(\frac{\rho_t^*}{z^b}\right)\right]\epsilon^2 + o(\epsilon^3) \tag{3.44}$$

For the second derivative term in the previous equation (3.44), we obtain

$$\partial_x^2\left(\frac{\rho_t^*}{z^b}\right) = \frac{(\partial_x^2\rho_t^*)(z^b)^2 + 2\rho_t^*(\partial_x z^b)^2}{(z^b)^3} - \frac{2(\partial_x\rho_t^*)(\partial_x z^b) + \rho_t^*(\partial_x^2 z^b)}{(z^b)^2} \tag{3.45}$$

and the corresponding formula for $\partial_y^2(\rho_t^*/z^b)$ is obtained by substituting y for x in Equation (3.45). Replacing the values of these second derivatives in Equation (3.44) yields

$$[(i)+(ii)+(iii)+(iv)] = o(\epsilon^3)+4\frac{\rho_t^*}{z^b}+\left[\frac{\Delta\rho_t^*}{z^b} + 2\rho_t^*\frac{\|\boldsymbol{\nabla} z^b\|^2}{(z^b)^3} - 2\frac{\boldsymbol{\nabla}\rho_t^*.\boldsymbol{\nabla} z^b}{(z^b)^2} - \rho_t^*\frac{\Delta z^b}{(z^b)^2}\right]\epsilon^2 \tag{3.46}$$

so that Equation (3.42) becomes

$$\rho_t^*(x,y) + \frac{\partial\rho_t^*(x,y)}{\partial t}\epsilon^2 + o(\epsilon^3) =$$
$$\left(1-\theta V_{x,y}\epsilon^2 + o(\epsilon^3)\right) \times \left[\rho_t^* + \left(\frac{\Delta\rho_t^*}{4} + \rho_t^*\frac{\|\boldsymbol{\nabla} z^b\|^2}{2(z^b)^2} - \frac{\boldsymbol{\nabla}\rho_t^*.\boldsymbol{\nabla} z^b}{2z^b} - \rho_t^*\frac{\Delta z^b}{4z^b}\right)\epsilon^2 + o(\epsilon^3)\right]$$

Keeping the terms up to the second order in ϵ provides

$$\begin{aligned}\frac{\partial \rho_t^*}{\partial t} &= -\theta V \rho_t^* + \frac{\Delta \rho_t^*}{4} + \rho_t^* \frac{\|\boldsymbol{\nabla} z^b\|^2}{2(z^b)^2} - \frac{\boldsymbol{\nabla}\rho_t^*.\boldsymbol{\nabla} z^b}{2z^b} - \underbrace{\rho_t^* \frac{\Delta z^b}{4z^b}}_{\rho_t^* \frac{\Delta z^b}{4z^b} - \rho_t^* \frac{\Delta z^b}{2z^b}} \\ &= \frac{\Delta \rho_t^*}{4} + \rho_t^* \left(\frac{\Delta z^b}{4z^b} - \theta V \right) - \frac{1}{2} \left(-\rho_t^* \frac{\|\boldsymbol{\nabla} z^b\|^2}{(z^b)^2} + \frac{\boldsymbol{\nabla}\rho_t^*.\boldsymbol{\nabla} z^b}{z^b} + \rho_t^* \frac{\Delta z^b}{z^b} \right)\end{aligned} \tag{3.47}$$

Noticing that $D = 1/(4\theta)$ and rewriting Equation (3.24) as

$$\frac{\Delta z^b}{4z^b} - \theta V = -\frac{\delta(x - x_b)\delta(y - y_b)}{z^b} \tag{3.48}$$

then combining this last result with

$$-\rho_t^* \frac{\|\boldsymbol{\nabla} z^b\|^2}{(z^b)^2} + \frac{\boldsymbol{\nabla}\rho_t^*.\boldsymbol{\nabla} z^b}{z^b} + \rho_t^* \frac{\Delta z^b}{z^b} = \operatorname{div}(\rho_t^* \boldsymbol{\nabla} \ln z^b), \tag{3.49}$$

we finally obtain for the optimal policy (3.47):

$$\begin{aligned}\frac{\partial \rho_t^*(x,y)}{\partial t} &= \frac{\Delta \rho_t^*(x,y)}{4} - \frac{1}{2}\operatorname{div}(\rho_t^*(x,y) \boldsymbol{\nabla} \ln z^b(x,y)) \\ &- \frac{\rho_t^*(x_b,y_b)\delta(x - x_b)\delta(y - y_b)}{z^b(x_b,y_b)}\end{aligned} \tag{3.50}$$

which corresponds to a Bloch diffusion equation with a drift (diffusion process subject to an external force) (Chaichian and Demichev, 2001) as in Equation (3.21). The initial condition at $t = 0$ is centered on the point of interest

$$\rho_0^*(x,y) = \delta(x - x_0)\delta(y - y_0) \tag{3.51}$$

Let us now look at the physical interpretation of the policy from Equation (3.50). The first term on the right-hand side of this equation, is a diffusion term with $\theta = 1$ or $D = 1/4$. The second term corresponds to a drift driven by the force $\mathbf{f} = \boldsymbol{\nabla} \ln z^b(x,y)$ (see Equation (3.21)) with $\gamma = 1/2$ and where $\ln z^b(x,y)$ plays the role of a **potential** known in advance (it is provided by the solution of Equation (3.24)). It must be noticed that this drift takes the direction of the shortest paths.

Eventually, the last term corresponds to an absorption (sink) of the probability density in the goal state. The role and interpretation of each component is summarized in the Table below.

Role	**Equation (3.50) components**
Diffusion term	$\dfrac{\Delta\rho_t^*(x,y)}{4}$
Drift	$\dfrac{1}{2}\text{div}(\rho_t^*(x,y)\boldsymbol{\nabla}\ln z^b(x,y))$
Absorption sink	$\dfrac{\rho_t^*(x_b,y_b)\delta(x-x_b)\delta(y-y_b)}{z^b(x_b,y_b)}$

3.4.5 Planning algorithm

This section summarizes the planning algorithm with the continuous-state version of the RSP framework. The algorithm is formally stated in Algorithm 7). This algorithm computes the optimal randomized policy at a point of interest (x_0, y_0) and a time t as follows:

1. Compute the backward variable $z^b(x,y)$ by solving Equation (3.24) with respect to $z^b(x,y)$ where $\ln z^b(x,y)$ is the associated potential.
2. Compute the optimal randomized policy $\rho_t^*(x,y)$, providing the probability of jumping to position (x,y) from position (x_0,y_0) during time step t, by solving Equation (3.50) with respect to $\rho_t^*(x,y)$ at the point of interest (x_0,y_0) and for the predefined length/time t.

Although this technique is stochastic, the best direction to follow corresponds to the orientation of the steepest ascent of $\ln z^b$, for which the gradient is maximum. Thus, the *deterministic* optimal policy tells us that we have to move in the direction of $\boldsymbol{\nabla}\ln z^b(x,y)$ at any point (x,y). This very simple rule

Algorithm 7 Planning with the continuous-state version of the RSP: Computation of the optimal policy at state (x, y).

```
Input:
    • $D > 0$: diffusion coefficient (or equivalently, $\theta$).
    • $n$: granularity of the squared grid (for solving with the finite difference method.
    • Boundary conditions.
Output:
    • $\rho_t^*(x, y)$: the optimal policy at state $(x, y)$.
1. Compute the backward variable $z^b(x, y)$ by solving Equation (3.24) with respect to $z^b(x, y)$.
2. Compute the optimal randomized policy $\rho_t^*(x, y)$, by solving Equation (3.50) with respect to $\rho_t^*(x, y)$ at the point of interest $(x_0, y_0)$ and for the predefined length/time $t$.
3. return $\rho_t^*(x, y)$
```

provides the minimal cost direction when θ is sufficiently large, and thus D is low – in which case the paths probability distribution is peaked on the shortest paths.

Interestingly, this policy is similar to an existing technique for mobile robot path planning (Schmidt and Neubauer, 1992; Schmidt and Azarm, 1992). The authors use a diffusion equation akin to Equation (3.24) (however, they assume a constant $V(x, y)$) and propose to follow the steepest ascent of the solution. They claim that this technique provides the shortest path to the goal state, but no proof is provided.

The complexity of this algorithm is $O(n^3)$ where n is the number of interior nodes in the grid. This is valid when applying the finite difference method, as a matrix inversion is needed.

The code of the planning algorithm is available at `http://github.com/silviagdiez/thesis`.

3.5 Simulations

In this section we present a series of simulations that have been performed in order to investigate the behavior of our method. The first series of simulations (see Figures (3.6–3.8)) illustrates the influence of the diffusion coefficient on a diffusion media with a simple Gaussian obstacle, and a single source and destination states. The second series of simulations (see Figures (3.9–3.11)) focus on solving a single-source multiple-destination maze for a fixed value of the diffusion coefficient.

These specific configurations for the simulation have been chosen as they have already been used in previous studies, and seemed like a good example for demonstration purposes. Any other configuration may have been possible as well. Although no numerical measures will be taken, the graphic representation of the results allow the reader to visually understand the results.

3.5.1 Methodology

For the **first simulation**, we assume a two-dimensional square plane with a source node (top-left corner) and a goal node (bottom-right corner). The agent has to move from the source to the destination while avoiding a Gaussian obstacle (situated in the middle of the plane, where red marks the top of the "mountain", and blue is the "valley").

In this case, several values of the diffusion coefficient have been investigated. In order to show the distribution of the paths, the average number of visits per node from Equation (3.30) has been plotted. Furthermore, to compute the forward and backward variables, a finite difference method has been used[3]. For each value of $D = \{1, 0.1, 0.01\}$, four results (graphics) are reported in a 40×40 grid of nodes, red being the highest probability of passing through the node, blue being the lowest:

1. The Gaussian obstacle.

[3]The granularity of the chosen grid was 40×40.

2. The logarithm of the forward variable computed from Equation (3.23).
3. The logarithm of the backward variable computed from Equation (3.24).
4. The average number of visits per node from Equation (3.30).

The **second simulation** takes place on a simple 36×36 maze (inspired from the one used by Dautenhahn and Cruse (1994)) with two obstacles of varying length. The objective, here, is to provide the agent with a plan, or policy, that will guide him through a maze with two walls (obstacles) from one source state to two possible goals. In this case, a constant value is chosen for the diffusion parameter, $D = 0.1$.

Similarly to the previous simulation, a finite difference approximation[4] has been used to compute the forward and backward variables. Once we have obtained these two variables we compute the average number of visits per node with Equation (3.30). Three different mazes have been used, where the obstacles change their length. The main difference among the first maze and the others, is that the second wall has no hole in it, leaving one single possible option of planning.

On the remaining two mazes, there is always the possibility of passing over the wall, or under it. For each maze, four results (graphics) are reported in a 36×36 grid of nodes, red being the highest probability of passing through the node, blue being the lowest:

1. The maze (in red), the source node (in yellow, on the left), and the two goal nodes (in yellow, on the right).
2. The logarithm of the forward variable computed from Equation (3.23).
3. The logarithm of the backward variable computed from Equation (3.24).
4. The average number of visits per node from Equation (3.30).

[4]The granularity of the chosen grid was 36×36.

3.5.2 Discussion of the results

Results of the **first simulation** are displayed in Figures (3.6–3.8), which show, as expected, an evident impact of D on the path chosen by the agent. Indeed, when D is high (low θ), the diffusion over the plane is important, and the Gaussian obstacle is thus explored. On the other hand, for low values of D (high θ), the shortest paths are favored, and the obstacle is completely avoided.

Results for the **second simulation** are shown in Figures (3.9–3.11). In this second case, the influence of the length of the obstacle determines the shortest path and, thus, the chosen goal node. It must be noted that when both goals are equidistant from the source node (see Figure (3.10)) both solution paths will be almost equally explored. On the remaining two cases (see Figures (3.9) and (3.11)), the shortest goal will be favored over the farthest one. It must be noted that, although this is a single-source multiple-destination problem, our method extends to the case of multiple-source multiple-destination problems. In order to add multiple sources and destinations, we will modify the Dirichlet boundary conditions, where a zero cost will be set for each source and destination, while the remaining boundary points will have an infinite cost.

3.6 Conclusion

In this chapter we have investigated the continuous-state counterpart of the discrete randomized shortest-path on a graph. It has allowed us to set some bridges between biased random walks on a discrete graph and the continuous-state Feynman-Kac diffusion process.

Furthermore, from an application point of view, it has provided an optimal randomized policy for solving continuous-state path planning problems with multiple sources and multiple goals. However, the main drawback of this model is that it assumes that paths are uncorrelated, which is hardly the case in practice. Further work will study the possibility of introducing a mass parameter (inertia) for smoothing the trajectories, therefore avoiding abrupt changes in direction. Indeed, the Wiener measure naturally accounts for a kinetic energy term cumulated along the trajectory (Chaichian and Demichev,

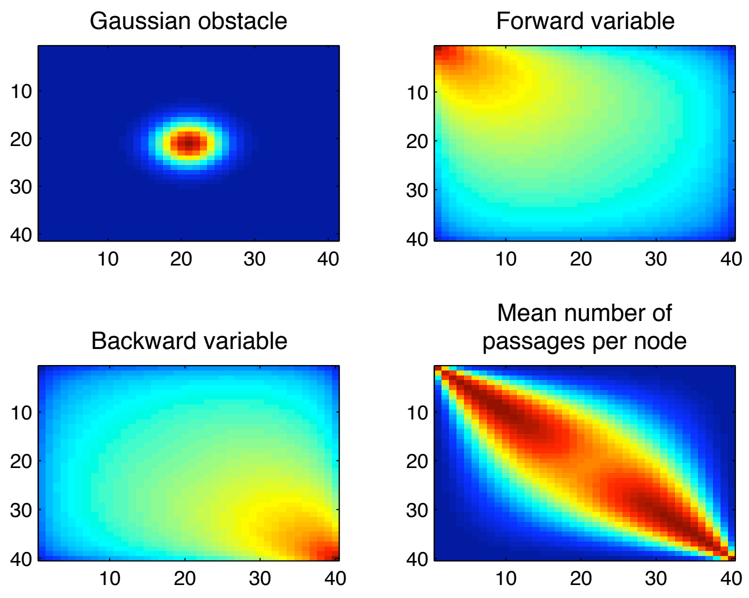


Figure 3.6: Continuous RSP with Gaussian obstacle and $D = 1$.

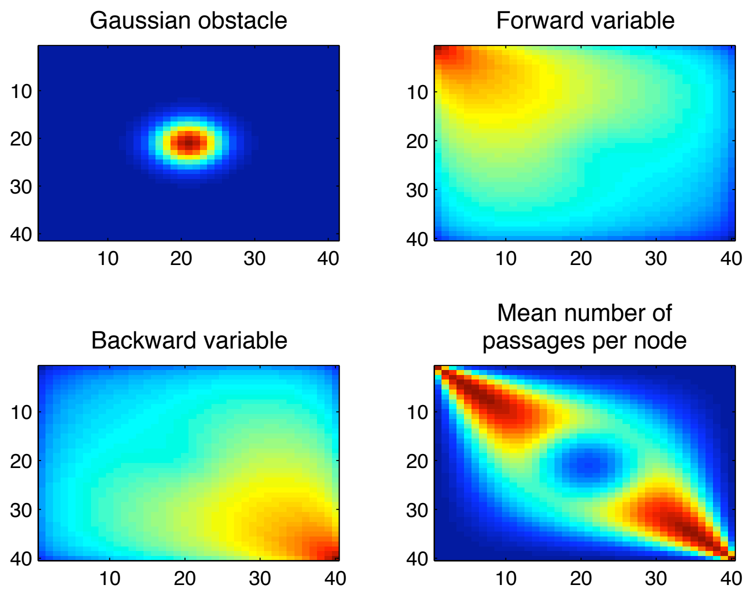


Figure 3.7: Continuous RSP with Gaussian obstacle and $D = 0.1$.

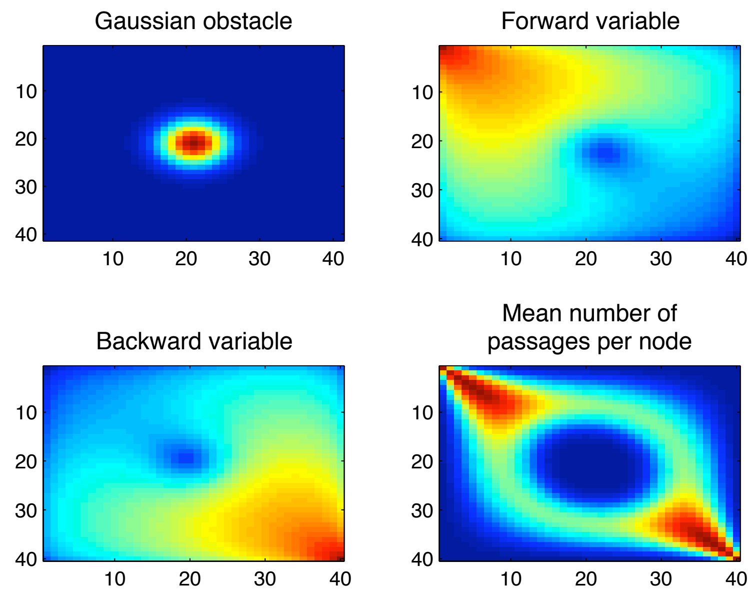


Figure 3.8: Continuous RSP with Gaussian obstacle and $D = 0.01$.

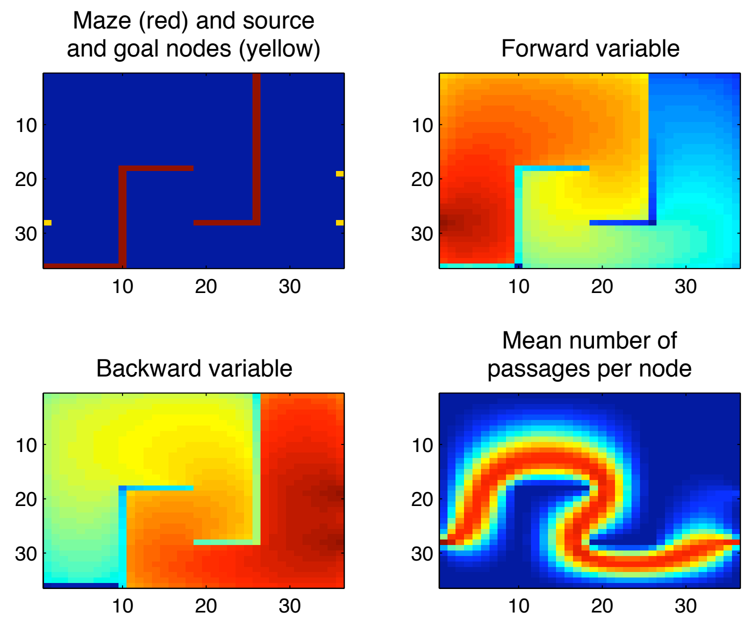


Figure 3.9: First maze with one source (on the left hand side) and two goal nodes (on the right hand side), with $D = 0.1$.

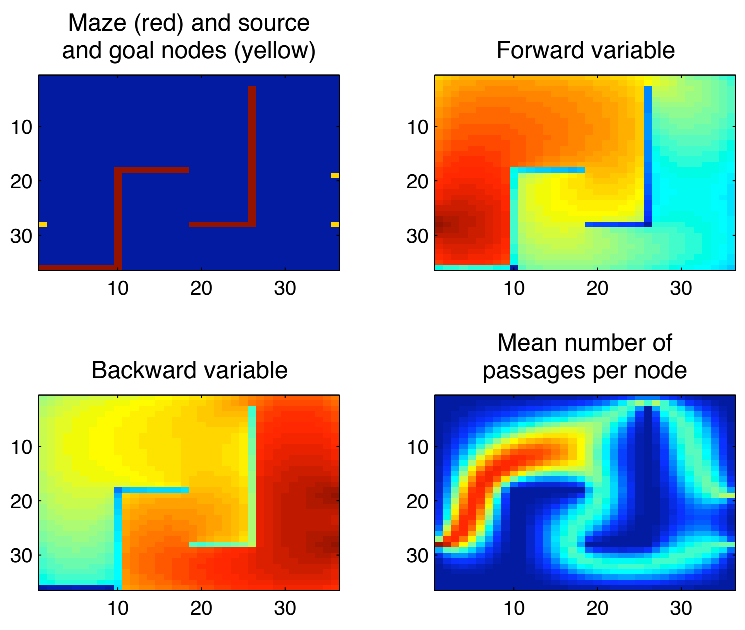


Figure 3.10: Second maze with the same source and goal nodes, with $D = 0.1$.

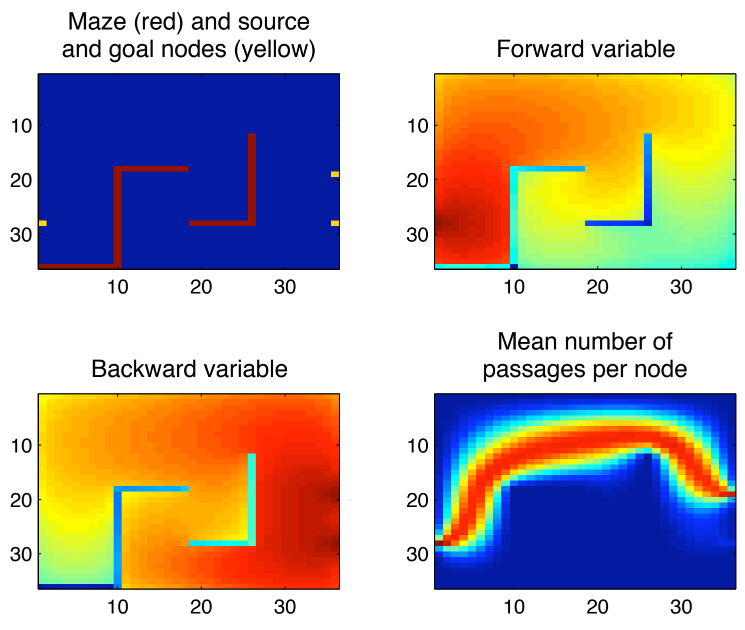


Figure 3.11: Third maze with the same source and goal nodes, with $D = 0.1$.

2001; Mazo, 2002). This would also enhance the physical interpretation of the model.

Finally, by discretizing the problem (see Section 3.5.1) and using a finite difference method, we are able to solve the planning problem by solving a system of linear equations instead of a more complex system involving partial derivatives. The finite difference method allows us to fix the granularity of the grid, being able to adapt it to the available computing resources. This could be useful when defining trajectories, e.g., on on-line video games, where there is a need for high-speed computation.

Chapter 4

The *R*minimax

Contents

Artificial intelligence (AI) techniques (Luger, 2009; Nilsson, 1998; Russell and Norvig, 2003) are widely used in realistic-behavior video games (Millington, 2006; Smed, 2006). These methods aim, e.g., at finding paths for motion planning, collaborating between computer entities, learning from past experience, proposing game strategies, etc.

The main focus of this chapter is on finding strategies for two-player perfect information zero-sum games (Morris, 1994; Osborne, 2004; Rasmusen, 1989), such as chess and draughts. These games can be seen as a succession of plays which alternate from one player to another, and where the profit is maximized for the current player – therefore, minimized for the opponent. They are often solved thanks to the well-known MINIMAX algorithm (Luger, 2009; Millington, 2006; Nilsson, 1998; Russell and Norvig, 2003; Smed, 2006) which is straightforwardly or indirectly used in most board games (see Section 4.1 for a more detailed introduction to MINIMAX).

From its inception, MINIMAX assumes perfect rationality for both players and, therefore, it is completely deterministic – the player will adopt the same deterministic strategy when encountering the same situation. This fact is illustrated by Figure 4.1 where we observe that player π_1 (X) follows the optimal strategy (marked in red) and reaches one of the winning states.

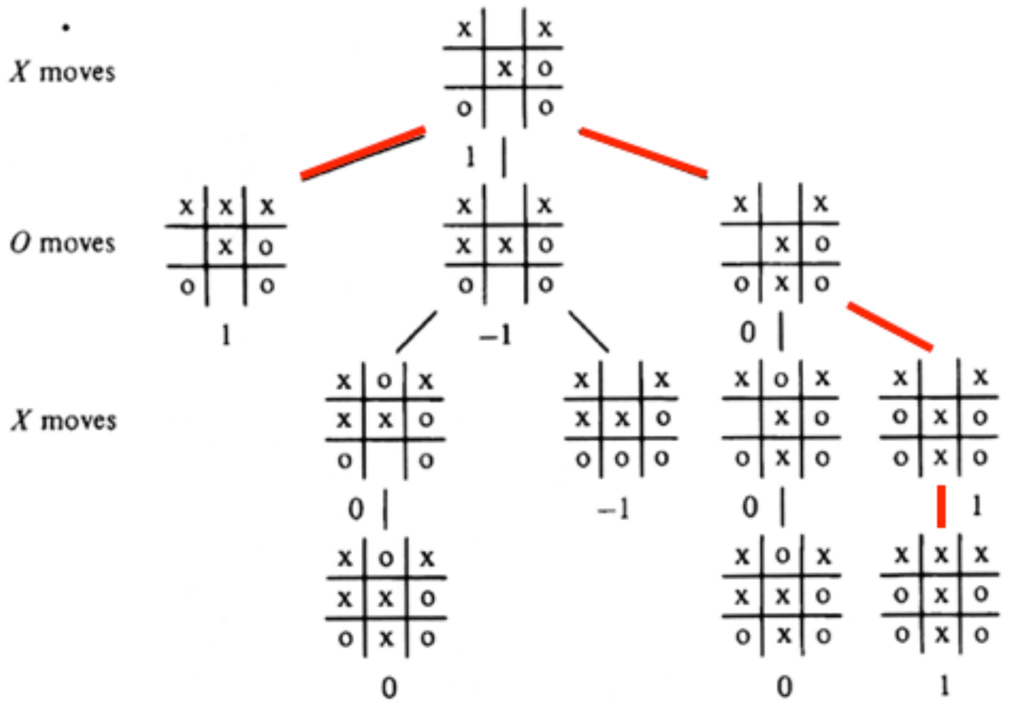


Figure 4.1: Optimal strategies for tic-tac-toe for π_1 (X) marked in red. Figure obtained from `http://mindfulintegrations.com/books/Technology/computer_science/algo/books/book9`.

Since the behavior of the AI player is completely predictable, the game might become annoying for the rival. Such problem is tackled in this chapter by proposing a simple way to randomize the strategy while still remaining optimal. The main idea is to control the spread randomness in the game tree, quantified through its Shannon entropy, and to select the optimal minimum expected-cost strategy for this entropy. In this way, good (low-cost) randomized strategies are favored, while bad ones (high-cost) are discarded.

The main difference between spreading the entropy at node level, i.e., taking a random decision using only the information of the current state, and spreading the entropy through all the game tree, i.e., taking a random decision using the information of the whole branch, is that the second case is more human-like. Indeed, people usually anticipate two or three moves (or more depending on the expertise of the player) before making a decision on their next move. This is what we try to mimic in this chapter.

This fact is illustrated in Figure 4.2 where a player of medium strength, π_1 (X), follows a sub-optimal strategy (marked in orange). Any of the sub-optimal strategies will lead him to a tie status (neither of the two players wins the game). On the other hand, by simulating a poor player, all strategies will lead him to losing statuses.

In order to adjust the strength of the player, i.e., the trade-off between exploitation and exploration of the game tree, we vary the entropy. In other words, our model aims at introducing/implementing bounded rationality (see Rubinstein (1998); Wolpert (2006) for a survey) to the MINIMAX algorithm. The proposed method, called $\mathcal{R}$minimax, is the application of the randomized shortest-path (RSP) framework by Saerens et al. (2009) (see also Section 1.1) to game trees.

The **research question of this chapter** is to investigate whether the RSP framework can be applied to zero-sum two-player games, to simulate an Artificial Intelligent player which has a more human-like behavior by spreading entropy through the whole game tree, and its comparison with similar algorithms. We present simulations run with our algorithm under different configurations for illustration purposes, and compare it with a similar algorithm to demonstrate its optimality.

Main contributions

This chapter provides a novel global optimal strategy based on the Sum-

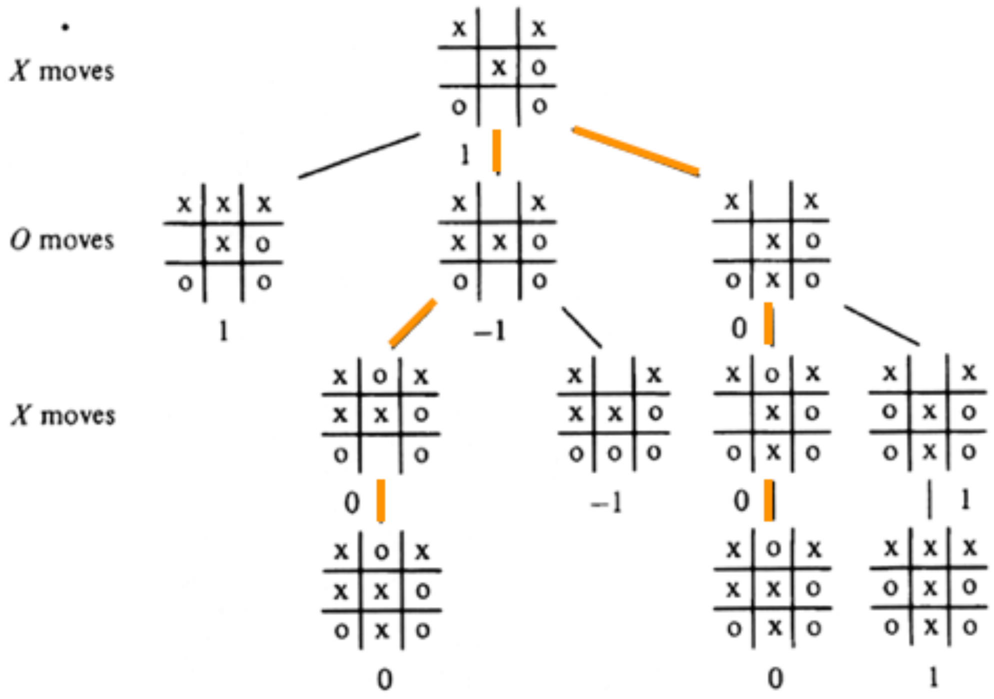


Figure 4.2: Sub-optimal strategies in tic-tac-toe for π_1 (X) marked in orange. Figure obtained from http://mindfulintegrations.com/books/Technology/computer_science/algo/books/book9.

over-Paths framework given a level of entropy for simulating the AI in two-player zero-sum games. Although the notion of entropy has been widely used for controlling randomness in AI, this new method spreads the entropy over full strategies, instead of single moves.

4.1 The MINIMAX algorithm

The MINIMAX algorithm (Luger, 2009; Millington, 2006; Nilsson, 1998; Russell and Norvig, 2003; Smed, 2006) computes the optimal strategy for two-player zero-sum games, provided that the opponent is fully rational, i.e., it will also play according to its optimal strategy.

In order to illustrate the MINIMAX principle, let us assume a game tree such as the one from Figure 4.3, a MAX player (π_1) and a MIN player (π_2) that want to maximize and minimize, respectively, their utility value (cost or reward). The utility value is initially defined for the leaf states of the game tree (winning states) as a low value for winning states of π_2 and higher values for winning states of π_1. Intermediate utility values indicate the advantage of one of the players over its opponent.

Once the leaf utility values are defined, the MINIMAX algorithm operates recursively on the game tree, iterating between the π_1 player which takes the maximum of its children's utility values (on odd-depth states) and the π_2 player which takes the minimum of its children's utility values (on even-depth states).

The first step of our example in Figure 4.3 shows the utility values for π_2 marked in blue – please note that we will always consider that π_1 is the player who moves first–. A utility value of 1 represents a winning state for π_1, -1 is a winning state for π_2, and 0 is a tie between both players. In the second step shown in Figure 4.4 we have propagated those values one level up (marked in red). The values remain the same, as the maximum of one value is the same value. On the third step in Figure 4.5, we propagate the values one level up, taking the minimum of all branches, as π_2 who plays min. For instance, the middle state is $\min\{0, -1\} = -1$. The final step in Figure 4.6 takes the maximum of all branches $\max\{1, -1, 0\} = 1$, as π_1 plays max. The two optimal trajectories for π_1 are marked in red in Figure 4.7.

4.2 Related work

As it is the nature of MINIMAX to search the whole game tree, much attention has been paid to reducing the search space. The simplest technique consists on bounding the depth of the tree with a *n-ply look-ahead* strategy (Luger, 2009), where n is the number of explored levels of the tree.

Also very common are the *alpha-beta* (AB) pruning techniques (Newell et al., 1958). The AB algorithm prunes irrelevant subtrees which will never

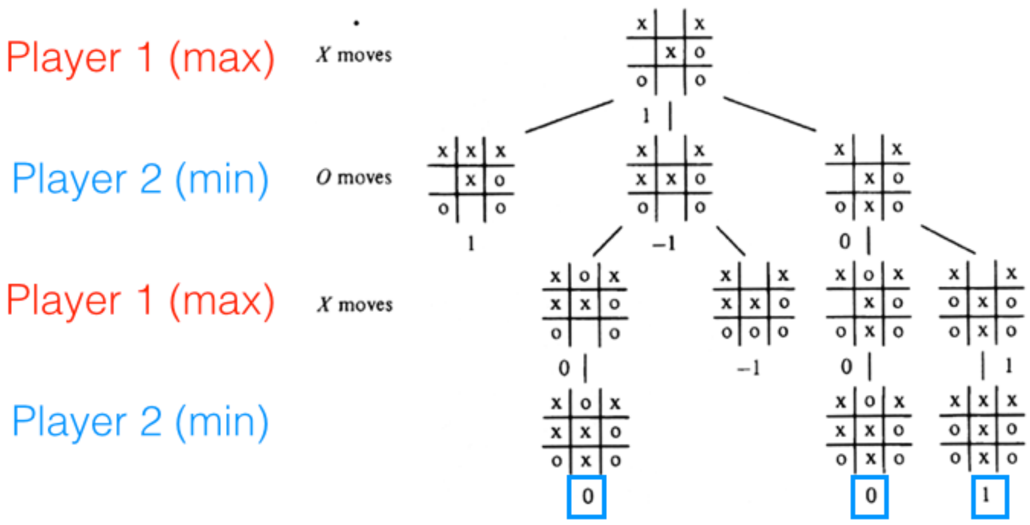


Figure 4.3: Example of the 1st step of a MINIMAX algorithm for a game tree of depth 4 and players π_1 (which plays max) and π_2 (which plays min). Utility values are shown below the status of the game.

be part of the MINIMAX strategy by using a window of two plies. An AB multi-player version is proposed by Sturtevant and Korf (2000). The *Negascout* algorithm introduced by Reinefeld et al. (1985) reduces even further this window size which allows to perform a faster pruning than AB. Nonetheless, the tree may be massively pruned leading to the elimination of good strategies. Similarly, the *Memory-enhanced Test Driver* (MTD(f)) (Plaat et al., 1995) limits the AB window size to zero. Although this pruning is faster, an initial “guess”, f, of the minimax position is required.

This method is also based on *transposition tables* which are used in games with a vast search space where recurrent states appear. In this case, it is more efficient to “remember” the decision taken the first time the state was observed than redoing the entire search. Despite that MTD outperforms the Negascout in the number of searched nodes, it suffers from stability issues, it depends on the transposition tables, and is also very sensitive to the initial guess. Eventually, a pruning technique which computes the expected value of the continued search is proposed by Russell and Wefald (1991). It has

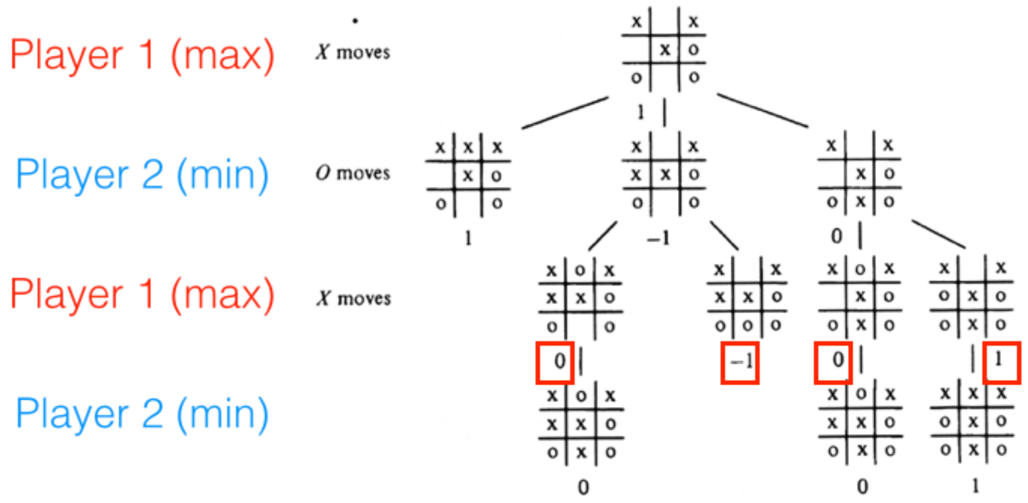


Figure 4.4: Example of the 2nd step of a MINIMAX algorithm for a game tree of depth 4 and players π_1 (which plays max) and π_2 (which plays min). Utility values are shown below the status of the game.

been shown by Russell and Wefald (1991) that this method suffers also from numerical instabilities.

Opening books (Buro, 1997) are another improvement technique applied to huge search space games. Efficient "opening" as well as "ending" game strategies that are often used by expert players are stored in these books. It is proved that initial strategies are critic for reducing the search space, as well as guiding the game towards the winning states. However, even when the search space is reduced, interaction time remains a key feature that must also be taken into consideration. *Iterative deepening* techniques may be useful in cases where the calculation time is unknown *a priori*. In this way, a strategy is available to interact at any time, but its quality will depend on the depth of the last explored tree. Often, this technique is used to choose a few good strategies obtained with a small depth and validated by extending them further. *Quiescence pruning* (Harris, 1975) avoids searching the branches of the tree whose heuristic function values are stable and, therefore, with no leadership changes.

MINIMAX has also been extended for chance games such as Backgammon.

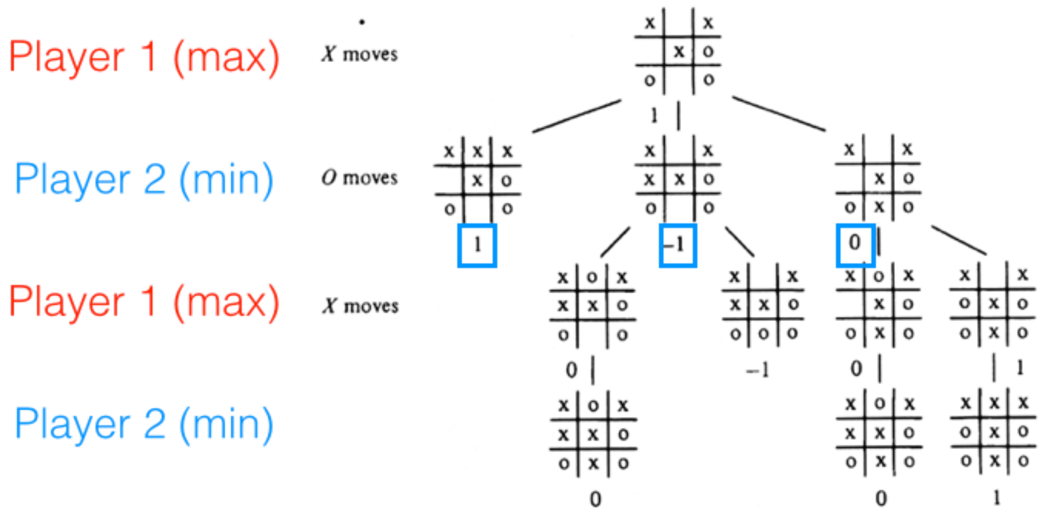


Figure 4.5: Example of the 3rd step of a MINIMAX algorithm for a game tree of depth 4 and players π_1 (which plays max) and π_2 (which plays min). Utility values are shown below the status of the game.

A version of the game tree with a new type of “chance” nodes representing the probabilistic states of the game (i.e., where a dice is thrown) is proposed by Michie (1966). Eventually, a stochastic approach which computes the probabilities of correctly scoring the following moves, via a heuristic function, is presented by Adelson-Velsky et al. (1988).

A different approach, involving randomized strategies, can be found in the Game Theory literature (Morris, 1994; Osborne, 2004; Rasmusen, 1989). *Mixed strategies* are an alternative to *pure strategies* in games where several decision makers interact in order to maximize their payoffs. Players must choose among a set of possible actions where each action has an associated cost or reward. In contrast to pure strategies where a player takes a deterministic action, $p_{\text{action}} = \{0, 1\}$, mixed strategies allow players taking an action with a given probability $p_{\text{action}} \in [0, 1]$. These probabilities are usually computed via the Nash equilibrium of the game, which corresponds to the best strategy (expected payoff) that player A can adopt while taking into consideration player B’s decision. Although the exact play remains unknown for the oppo-

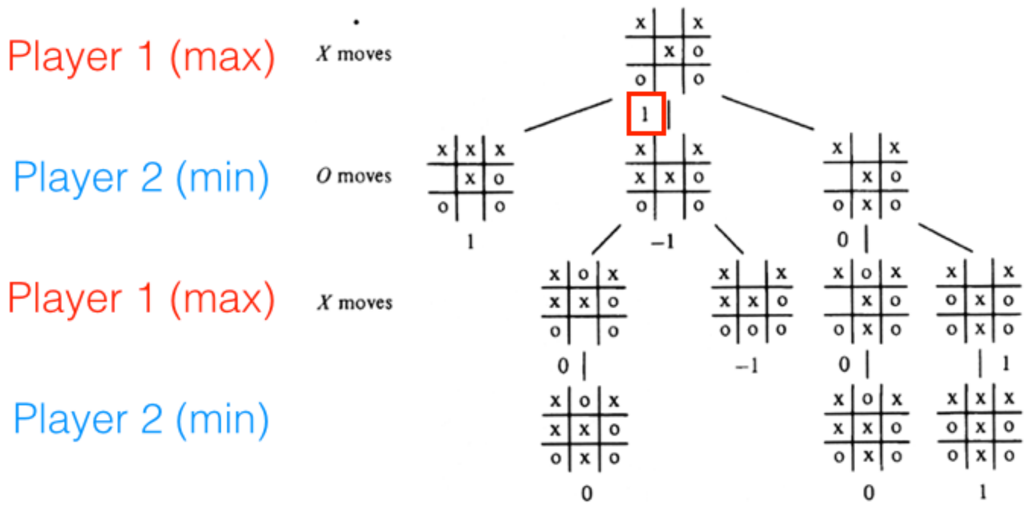


Figure 4.6: Example of the 4th step of a MINIMAX algorithm for a game tree of depth 4 and players π_1 (which plays max) and π_2 (which plays min). Utility values are shown below the status of the game.

nent, the probabilities of his actions are known in advance, letting the game be pseudo-random. An extension of these strategies for two-player turn-based random games are *stochastic games* (Shapley, 1953). This technique tries to maximize the expected payoff for a player by choosing an optimal strategy and its computation has been the subject of several studies such as the ones by Patek and Bertsekas (1997); Somla (2005).

Nevertheless, little attention has been paid to modeling the strength of an adversary in two-player zero-sum games in the artificial intelligence (AI) community. A basic approach consists in using the n-ply look-ahead algorithm was presented by Luger (2009) in order to vary the capacity of a rival. Unfortunately, n may be tough to tune as it depends on the game and the branching factor. I.e., for low values of n (i.e., in chess a small $n < 6$), the AI-based opponent can easily be outperformed by the user (who normally plans 6 or 8 plies ahead), while for very high values ($n > 8$) it may become extremely difficult to win. Other frequently used techniques are ϵ-*greedy* (Sutton and Barto, 1998), where the optimal branch is taken with probability $1 - \epsilon$ and

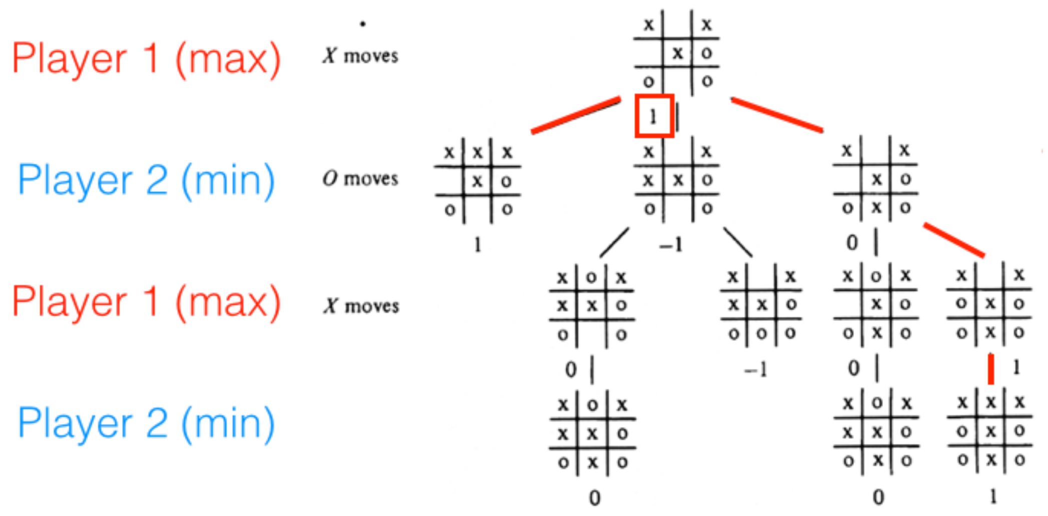


Figure 4.7: Optimal strategies for π_1 marked in red.

a random branch with uniformly distributed probability $\frac{\epsilon}{\text{number\_branches}}$, *Boltzmann exploration* (Sutton and Barto, 1998), where the probability of taking a branch follows a Boltzmann distribution with an inverse temperature which depends on the state-specific exploration coefficient, and techniques based on the addition of noise to the evaluation function. However, such techniques focus on the current state, limiting their strategy to local decisions and failing to find an optimal global strategy over the whole game tree for a given entropy (Achbany et al., 2008).

This idea of *bounded rationality* (see the work, e.g., Gigerenzer and Selten (2002); Rubinstein (1998); Wolpert (2006)) has already been applied in a large number of fields from Psychology (Simon, 1991) to Artificial Intelligence (Lee and Wolpert, 2004). In this last category, we find the work from Wolpert (2006), where the link between game theory and statistical physics is analysed. In this context, he shows how Shannon's information theory provides a framework for bounded rational game theory. In particular, when we know both players' mixed strategy and their expected cost, the probability distribution of possible actions should follow a Boltzmann distribution. However, the author does not provide a precise algorithm implementation for his ideas. This chapter can be considered as a concrete instantiation of these ideas for two-player zero-sum games. Nested Monte-Carlo search (Cazenave, 2009) provides another bounded rational algorithm over a game tree, which combines nested calls with randomness and memorization of the best sequence of moves.

The proposed approach of this chapter does not only focus on modeling the strength of an adversary, but also on ameliorating the AI of the MINIMAX by adding probabilistic, more human-like, while still optimal, strategies.

4.3 A randomized MINIMAX

As already mentioned, MINIMAX has been widely applied for emulating an opponent in two-player zero-sum games. While being very useful in most situations, it, however, suffers from some drawbacks. Firstly, the assumption of perfect rationality for both players is unrealistic, as human players often incur into error. Secondly, it does not address the issue of vast search space for certain games, and therefore, the use of a heuristic function is often necessary, which is usually hard to define. Thirdly, the behavior of the player is deterministic and thus predictable. Fourthly, in its basic form, controlling the strength of the player is not feasible. The developed approach of this section overcomes some of these shortcomings.
It can be observed from the game tree that a deterministic strategy leads to a path from the root node (initial state) to a leaf node (end game – winning/losing state). MINIMAX chooses the path which maximizes the gain of the current player, while minimizing the gain of the adversary. A variant of MINIMAX which will randomize the choice among all possible paths of the game tree is introduced. The advantage of this technique is fourfold

1. deterministic strategies are avoided, therefore eliminating the predictability of the game;
2. perfect rationality of the player is not assumed;
3. control over the strength of the player is allowed;

4. the method is still optimal in a certain sense.

Although the issue of the search space is not tackled in this chapter, as for MINIMAX, any of the existing techniques could be applied in order to reduce the size of the explored tree.

4.4 The $\mathcal{R}$minimax algorithm

The application of the RSP framework (see Section 1.1 for more details) to the game tree will allow to bias the transition probabilities towards better or worse solutions as θ increases or decreases. In that case, the graph G is a tree and it is therefore acyclic. Equation (1.19) defines therefore the recurrence relation allowing to compute the backward variables z_k^b from the destination node n to each intermediary node k.

Assume that π_1 is our AI player, and π_2 is the opponent. We will randomize π_1's strategy while still assuming that π_2 plays rationally. The set of winning/losing states indicating the end of the game will be denoted by $\mathcal{N}$ and the set of paths is now $\mathcal{P}_{1\mathcal{N}}$. By applying the RSP framework to this situation, the backward variables (Equation (1.19)) are redefined in terms of the following recurrence relations (please note that each final state is an absorbing node):

$$\begin{cases} z_k^b & = 1, \text{ for } k \in \mathcal{N} \\ z_k^b & = \begin{cases} \displaystyle\sum_{k' \in \text{Succ}(k)} \exp[-\theta c_{kk'}]\, z_{k'}^b, \text{ if } k \text{ is in } \pi_1\text{'s turn} \\ \displaystyle\min_{k' \in \text{Succ}(k)} \exp[-\theta c_{kk'}]\, z_{k'}^b, \text{ if } k \text{ is in } \pi_2\text{'s turn} \end{cases} \end{cases} \tag{4.1}$$

where $k \notin \mathcal{N}$ is assumed. It can be observed that when π_1 (the AI player) plays, it takes into account the costs of all successors of state k for randomizing its future strategy, while π_2 plays the best strategy (most profitable to him) by considering only one branch of the tree – the most promising one.

Indeed, since the transition probabilities (the policy followed by player π_1)

are proportional to $\exp[-\theta c_{kk'}]\, z^b_{k'}$ (see Equation (1.16))[1] , according to Equation (4.1), π_2 chooses the action corresponding to the lowest (min) transition probability, i.e. the move that is *least favorable* to his opponent π_1, all the other moves being dismissed – the game tree is pruned accordingly.

As π_1 and π_2 play in turn, the value of the backward variables is computed by alternating both equations. It must also be noticed that in order to avoid overflow or underflow problems, the standard formula for the logarithm of a sum (see, e.g., Huang et al. (1990)) can be applied when computing z^b_k.

Although it is not immediately obvious from Equation (4.1), player π_1 *minimizes* the expected cost to the end-game by following the optimal policy provided by Equation (1.16) (subject to entropy constraints – this directly follows from the RSP framework, see Equation (1.19)) while player π_2 tries to *maximize* it. Indeed, let us take $-\frac{1}{\theta}\log$ of each member of the recurrence relation for player π_2 in Equation (4.1),

$$-\frac{1}{\theta}\log(z^b_k) = -\frac{1}{\theta}\log\left(\min_{k'\in\mathrm{Succ}(k)}\left[\exp[-\theta c_{kk'}]z^b_{k'}\right]\right) \tag{4.2}$$

By using

$$-\log(\min(x,y)) = \max(-\log(x), -\log(y)) \tag{4.3}$$

and defining

$$v_k = -\frac{1}{\theta}\log(z^b_k) \tag{4.4}$$

where v_k can be interpreted as a potential (see Section 3.4.1), we obtain for player π_2

$$v_k = \begin{cases} 0 & \text{for } k \in \mathcal{N} \\ \max\limits_{k'\in\mathrm{Succ}(k)}(c_{kk'} + v_{k'}) & \text{if } k \text{ occurs during } \pi_2\text{'s turn} \end{cases} \tag{4.5}$$

Now, this is exactly the recurrence equation, akin to the Bellman equation (see, e.g., Bertsekas (2000)), allowing to compute the maximal-cost path to

[1]Here, as in the "Sum-over-paths edit distance" chapter, we omit $p^0_{kk'}$ which is constant in a given state of the game, each decision or move being equiprobable when playing at random.

the end-game states[2]. Therefore, player π_2 consistently tries to maximize the cost. If player π_1 would only consider the best move, as does player π_2, we recover the standard minimax.

Notice that the strategy of player π_1 in Equation (4.1) is also closely related to the Bellman-Ford algorithm (see Francoisse et al. (2013)). Indeed, by defining v_k as in Equation (4.4), we obtain from Equation (4.1),

$$\begin{aligned} v_k &= -\frac{1}{\theta} \log \sum_{k' \in \text{Succ}(k)} \exp[-\theta c_{kk'}]\, z^b_{k'} \\ &= -\frac{1}{\theta} \log \sum_{k' \in \text{Succ}(k)} \exp[-\theta c_{kk'}]\, \exp[-\theta v_{k'}] \\ &= -\frac{1}{\theta} \log \sum_{k' \in \text{Succ}(k)} \exp[-\theta (c_{kk'} + v_{k'})] \end{aligned} \tag{4.6}$$

The function $f(x_1, x_2, \ldots; \theta) = -(1/\theta) \log \sum_k \exp[-\theta x_k]$ is called the *soft minimum* (Cook, 2011). It is a kind of smoothed minimum between the different x_k, hence its name of soft minimum. When $\theta \to \infty$, it converges to the minimum of the x_k so that the Equation (4.6) reduces to the Bellman-Ford formula (taking the minimum over k' of the $(c_{kk'} + v_{k'})$). When $\theta < \infty$, it defines an extension of Bellman-Ford taking all paths into account and favoring shorter paths.

During the game, once the backward variables have been computed, the AI player π_1 chooses his next move according to Equation (1.16), that is, in state k, he selects k' with probability $z^b_{k'} \exp[-\theta c_{kk'}] / \sum_{l \in \text{Succ}(k)} z^b_l \exp[-\theta c_{kl}]$.

The $\mathcal{R}$minimax algorithm is summarized in Algorithm 8 and the code is available at `http://github.com/silviagdiez/thesis`. Note that, when θ takes a high value, near-optimal strategies are chosen by the AI player π_1, while for small values, he will model a weak rival with a poor strategy. As an example of the effect of the different θ on the transition probabilities, let us consider the following case: assume a trivial binary game tree with only three levels where the current node is the root node, and the aim is to reach a winning

[2]For a study of the relationships between the randomized shortest-path policy and the Bellman-Ford algorithm, see (Francoisse et al., 2013).

node – associated to a reward (see the simulation methodology in Section 4.5 for more details) – while playing with a strength θ. The cost of each play is +1. Once all quantities have been computed, the results shown in Table 4.1 are obtained.

Algorithm 8 $\mathcal{R}$minimax: computation of the transition probabilities.

```
Input:
    • $G$: The generated game tree obtained with the MINIMAX algorithm.
      The root of the game is $k \in \pi_1$.
    • $\theta > 0$: The degree of randomization of the tree ($\infty$ for a perfect rational
      player, $\simeq 0$ for an almost completely random player).
    • $c_{kk'} \geq 0$: The cost of each arc of the tree.
Output:
    • $p_{kk'}$: transition probabilities for the next play.
1. Assign $z^b_n = 1$ for each $n \in \mathcal{N}$.
2. Compute recursively the $z^b_k$ according to Equation (4.1).
3. Compute the corresponding $p_{kk'}$ according to Equation (1.16).
4. return $p_{kk'}$: the transition probabilities for the next play.
```

The complexity of the algorithm is $O(b^d)$ where b is the average branching factor, and d is the depth of the tree.

It must be noticed that, when θ is large, the optimal strategy given by the MINIMAX algorithm is recovered. As θ decreases, the transition probabilities are less biased towards the optimal solution. In the case of $\theta \to 0$, the assigned costs become irrelevant, and therefore the strategy is utterly random (the transition probabilities p_{12} and p_{13} are almost uniformly distributed).

	$\theta = 3$	$\theta = 0.5$	$\theta = 0.001$
p_{12}	0.998	0.728	0.5001
p_{13}	0.002	0.268	0.4999

Table 4.1: Example of transition probabilities p_{ij} (transition probability from node i to child j) for a simple binary game tree of depth 2, when varying θ.

Link with Reinforcement learning techniques

Reinforcement learning, according to Sutton and Barto (1998) is "learning what to do – how to map situations to actions – so as to maximize a numerical reward signal. The learner is not told which actions to take [...] but instead must discover which actions yield the most reward by trying them". As the effects of actions cannot be fully predicted, the agent must keep exploring his environment. Therefore, the balance between exploration and exploitation is one of the major challenges of reinforcement learning.

The main elements that are involved in reinforcement learning are:

1. a *policy* that indicates how the agent behaves in a certain state and time, and which is usually stochastic;

2. a *reward function* that maps a state into a value that indicates the interest of that state, and which can be stochastic;

3. a *value function* which accumulates the individual rewards that an agent can obtain when starting in a given state, and indicates a measure of the long-term interest of a state;

4. a *model of the environment* which predicts what will happen if an action is taken on a given state.

Reinforcement learning techniques are of interest when the rewards of an agent are stochastic rather than deterministic, where supervised learning techniques are usually applied. The main difference between both methods is that reinforcement learning evaluates actions, while supervised learning searches the parameter space of a given model.

These techniques specify how an agent must modify its policy as a result of previous experience of taking actions in given states. Two well-known techniques are the ϵ-greedy which chooses actions with equal probability, and the *Softmax* (also called *Boltzmann exploration*) which varies the action probabilities as a graded function of their estimated value. As stated by Thrun (1998), Boltzmann distributions provide a way to combine random exploration with exploitation, and the likelihood of picking an action is exponentially weighted by its utility.

Although similar approaches (but not optimal) can be found in reinforcement learning (Carmel and Markovitch, 1999; John, 1994; Singh et al., 2000; Sutton and Barto, 1998), to the better of our knowledge, no technique spreads the probability over full branches of a tree. The main difference between reinforcement learning techniques and the RSP is that the exploration carried out by the latter is monitored optimally, while is not the case of the first. However, in reinforcement learning the next state is randomly chosen, while it is deterministic with the RSP, i.e., the entropy is spread over a set of known states.

Link with Monte-Carlo tree search techniques

The RSP frameworks is also related to Monte-Carlo tree search techniques (MCTS) (Chaslot, 2010). MCTS are a best-first search method which is guided by consecutive Monte-Carlo simulations. The advantage of MCTS regarding *alpha-beta* (AB) techniques is that it does not rely on an evaluation function on a given position. Instead, it uses the knowledge obtained by Monte-Carlo simulations, e.g., in the game of Go where no such reliable evaluation function is available. MCTS simulations consists on progressively refined random moves. Each node contains its current value, which is an average of the results of all simulations, and its visit count. With this information we can control the balance between the exploration and exploitation of the game tree.

The objective of the MCTS is to find the best strategy possible by following these four steps:

1. Selection step: the game tree is traversed from root to one of its leaf nodes. During this step we can control the balance between exploration and exploitation by deciding to explore a node with low current value, and which has not been very visited.

2. Expansion step: a new node is added to the game tree.

3. Simulation step: all moves are played until the end of the game from that new node.

4. Back-propagation step: the results from the simulation are propagated

until the added node.

This trade-off between exploration and exploitation has proven to be useful in Multi-Armed Bandit problems (Robbins, 1952), where a player wants to maximize his reward by choosing one of the multiple arms of a gambling device. Boltzmann exploration is also used as bandit strategy in order to avoid the lookahead-pathology, and shows competitive results (Kocsis and Szepesvári, 2006). Furthermore, the bandit based method from Coquelin and Munos (2007) performs efficient "cuts" of sub-optimal branches with high confidence, by allowing to control this trade-off.

Chatriot et al. (2008) propose the application to the game of Go of a bandit technique which biases the exploration of the tree in order to find the most suitable strategy to be explored according to previous information (such as the number of times a state has been explored, or the number of times that it led to a victory). Yet, most of these techniques eventually find an optimal policy and stop exploring the graph, therefore loosing their stochastic behavior. On the other hand, John (1994) proposes a reinforcement learning technique which continually explores the graph. However, the convergence to the optimal policy can no longer be proved.

In summary, the $\mathcal{R}$minimax contributions are:

1. to model non-rational players,
2. to control the strength of a player,
3. to avoid the total predictability of a player.

4.5 Simulations

In order to illustrate the proposed method, systematic simulations on two-player zero-sum games have been performed. Two common well-known games such as *Tic-Tac-Toe* and *Connect-4* are tested.

The Tic-Tac-Toe is a popular game that takes place on an horizontal 3×3 grid where two players position a token (circle or cross) alternatively. The aim of the game is to be the first to place 3 consecutive tokens in a row, column or diagonal. The aim of the Connect-4 game is similar, although the grid is 4×4, and it is placed in a vertical position where the players drop their colored discs alternatively, letting the bottom rows be filled first. We have chosen these two games, as they are well-known but represent a different degree of difficulty and search space. We did not want to tackle chess, as this is a highly complex game – whose simulation would have taken longer – for illustration purposes, but it could have been possible.

Game trees for both games have been generated with both the MINIMAX as well as the alpha-beta (AB) algorithm. Two AI opponents have been simulated, each with a different strength, θ_i, for testing the behavior of our method when confronting different heterogeneous players. The simulation methodology is as follows:

1. The game tree for the current node, k, is computed with the MINIMAX algorithm:
   1.1. the tree can be either fully generated, or limited to a 5-ply lookahead (depth = 5)[3].
   1.2. the tree can be be pruned with the alpha-beta algorithm.
2. A reward is assigned to each transition to a winning node. Cost are computed as follows:
   2.1. in the case of a full game tree, lower costs are assigned to winning nodes and higher ones to losing nodes. Tie nodes are assigned a value between those of winning and losing nodes.
   2.2. in the case of a 5-ply lookahead, the heuristic described in Section 4.5.2 is used.

[3]It must be noted that, at this stage, any of the previously explained techniques for reducing the search space could be applied (transposition tables, pruning techniques, etc.). However, only the case of pruning is showed here, as our purpose is merely illustrative.

2.3. all other internal arcs are assigned a cost of $c_{kk'} = 1$ so that short-winning paths will be preferred to long-winning paths. It is the length of the path and the final transitions' costs that matter when choosing a certain strategy.

3. For both players, apply the $\mathcal{R}$minimax algorithm as described in Algorithm 8 with strength θ_i for player i. This allows to compute the biased transition probabilities.

4. Choose the next state k' among all successor of k with probability $p_{kk'}$.

5. If k' is a winning/losing end-game state, the result is increased/decreased by one unit according to the winner and a new game is started. Otherwise it is the next player's turn and return to step 1.

This whole procedure is repeated 100 times (different runs) and returns a result r which takes its values in $r \in [-100, 100]$, which indicates the number of victories of both players. If $r > 0$, player π_1 has $|r|$ out of 100 victories over π_2. Otherwise, the winner will be π_2 with $|r|$ victories out of 100, and $r = 0$ represents result parity. We report numbers of victories, as it is easy to correlate them with the strength of the player, i.e., θ.

Please note that the values of θ have been chosen in a logarithmic scale so to investigate the influence of the parameter in the performance of players (similar values of θ were not representative enough).

4.5.1 $\mathcal{R}$minimax with full game tree

For getting a better insight about $\mathcal{R}$minimax's behavior, it is first applied to Tic-Tac-Toe on the full game tree generated by the MINIMAX algorithm. In order to visualize the performance of our method when two players of different strengths interact, 100 runs have been performed between two players of varying strength θ. According to our simulation methodology stated above, the performance of both players is recorded when applying the $\mathcal{R}$minimax. Tested values of θ are $\theta_1 = \theta_2 = \{0.1, 0.5, 1, 5, 10\}$. The resulting curves are shown in Figure 4.8.

As it can be observed, all curves have a similar shape but start at different levels. This can be translated into a high resemblance in the behavior of the AI players: when $\theta_1 >> \theta_2$, player 1 wins, while for $\theta_1 << \theta_2$, it is player 2 who leads the game. Such behavior fulfills what we expected as for $\theta \to \infty$, the entropy is 0 and thus the player chooses an optimal strategy and *vice versa.* In the case of $\theta = \infty$, the game reduces to the MINIMAX strategy. The level at which a curve begins depends on the difference between both θ's.

On the other hand, the slope of the curves reflect the effect of the relative advantage of π_1 over π_2. Indeed, π_1 always moves first, and it has an advantage over π_2. This can be observed in Figure 4.8 where a lower slope is shown for low values of θ_2 for π_2.

4.5.2 $\mathcal{R}$minimax with 5-ply lookahead and heuristics

Another frequent tool used in AI are heuristics and evaluation functions (see, e.g., Russell and Norvig (2003)). The performance of our method when using a partial game tree combined with the use of heuristics is studied in this section. In this experiment, the investigated game is Connect-4.

As generating the full game tree would be computationally expensive, a 5-ply lookahead method is here implemented and combined with the use of a heuristic function for scoring the final transitions. The applied heuristic is the one proposed by Stenmark (2005), and corresponds to the sum of two quantities: the number of winning lines that may still be done for each following move, plus a fixed quantity which corresponds to the goodness of the empty positions which are left (some positions are more versatile than others). Tested values of θ are $\theta_1 = \{0.01, 0.1, 0.7, 10\}$ and $\theta_2 = \{0.01, 0.1, 0.5, 1, 2, 5, 10\}$. Results are shown in Figure 4.9.

These resulting curves and the ones of the previous section are alike. Yet, as the game tree is limited to a certain depth and Connect-4 has a wider set of initial positions than Tic-Tac-Toe, the relative advantage of π_1 is not as clear as in the former case. Indeed, for observing the same effect, significantly lower values of θ are needed (than those of the Tic-Tac-Toe).

4.5.3 $\mathcal{R}$minimax with 5-ply lookahead and alpha-beta pruning

For this simulation, a partial game tree of depth 5 has been generated for the game of Connect-4. This time, a pruning algorithm reducing the search space is applied. The objective is to observe the behavior of the $\mathcal{R}$minimax algorithm combined with a technique which reduces not only the depth of the tree, but also the search space. Tested values of θ are $\theta_1 = \{0.01, 0.1, 0.3, 0.7, 10\}$ and $\theta_2 = \{0.1, 0.2, 0.5, 1, 2, 5, 10\}$. Results are shown in Figure 4.10.

These results are consistent with those of previous sections. However, in this case, the relative advantage of π_1 is even smaller. In contrast to the former case, a smaller θ_2 allows π_2 winning as θ_1 decreases. This is due to the pruning of the alpha-beta, as it restrains the set of explored branches.

4.5.4 Comparison with ϵ-greedy

Although the $\mathcal{R}$minimax is proved to be optimal for a fixed entropy, this section illustrates its optimality when compared with other popular bounded-rational algorithm: the ϵ-greedy (Sutton and Barto, 1998).

In order to illustrate the behavior of both algorithms, two game trees have been used (see Figures 4.11 and 4.12). As the ϵ-greedy algorithm makes local decisions (at a state level) and the $\mathcal{R}$minimax makes strategic decisions (at a path level), a fixed entropy for both algorithms has been fixed on the tree, so that the expectation of the cost can be compared under similar conditions.

In order to estimate both $\hat{\theta}$ and $\hat{\epsilon}$, a binary search algorithm has been used as follows:

1. The value of the entropy over the tree H_0 is fixed.

2. Two initial values of θ (or ϵ) are used to compute the probabilities of the different paths or strategies, $\wp$, of the tree (see Algorithm 8).

3. The obtained entropy of the tree H_t is computed thanks to Equation 1.4.

4. If the obtained $H_t = H_0$ we consider the value of θ as $\hat{\theta}$ (or respectively ϵ).

5. Otherwise we apply the binary search and continue searching on a sub-interval of the initial values until $H_t = H_0$.

Table 4.2 shows the result of this experiment for the tree from Figure 4.11 with utility values $\{1, 2, 98, 99, 100\}$. Each arc is supposed to have a unit cost $c_{kk'} = 1$. For different levels of the entropy (the constraint of our optimization problem), parameters are estimated ($\hat{\theta}$ and $\hat{\epsilon}$), letting us compute the expected cost (the quantity to optimize) when following a $\mathcal{R}$minimax or ϵ-greedy strategy.

The implemented $\mathcal{R}$minimax corresponds with Algorithm 8, and the implementation of the ϵ-greedy combines the typical ϵ-greedy[4] for π_1 (odd levels) with a standard MINIMAX for π_2 (even-levels). The same experiment has been repeated with the tree from Figure 4.12 with utility values $\{1, 2, 3, 4, 5\}$ in order to analyze the impact of the chosen utility values on the comparison between both algorithms. Results are presented in Table 4.3.

Results show that the expected cost in the case of the $\mathcal{R}$minimax is always below those values for the ϵ-greedy regardless of the utility values. This behavior was expected, as the $\mathcal{R}$minimax is indeed optimal for a given entropy. It must also be noted that the more they differ the utility values, i.e., the quality of the goals, the bigger the difference between the expected cost of both algorithms.

[4]Let us remind that ϵ-greedy takes the optimal action with probability (ϵ/n) and other actions with probability (1-ϵ)/m, where n is the number of optimal actions, and m is the number of non-optimal actions.

H	$\hat{\theta}$	$\mathbb{E}[C]_{Rminimax}$	$\hat{\epsilon}$	$\mathbb{E}[C]_{\epsilon-greedy}$
1.0	0.4469	1.9000	0.6679	12.5116
0.9	0.6397	1.7150	0.7187	10.7515
0.8	0.7984	1.5766	0.7665	9.0930
0.7	0.9474	1.4624	0.8080	7.6545
0.6	1.0987	1.3636	0.8456	6.3514
0.5	1.2574	1.2785	0.8798	5.1667
0.4	1.4356	1.2035	0.9106	4.1005
0.3	1.6456	1.1385	0.9384	3.1359
0.2	1.9239	1.0818	0.9628	2.2897
0.1	2.3634	1.0348	0.9838	1.5619

Table 4.2: Comparison of $\mathcal{R}$minimax and ϵ-greedy algorithms on the tree from Figure 4.11 for different levels of entropy H. Parameter estimations ($\hat{\theta}$ and $\hat{\epsilon}$) as well as the expected costs are reported.

4.6 Conclusion

In this chapter we have presented a randomized version of the MINIMAX algorithm which turns a zero-sum perfect-information two-player game into a non-deterministic game adapted to the player's level. By using the randomized shortest-path framework, it is not only able to compute the probabilities of each play through dynamic programming techniques, but also to optimally vary the strength of the AI by adjusting the entropy through the θ parameter. There is a clear relation between the $\mathcal{R}$minimax algorithm and mixed strategies in game theory and the methods used in reinforcement learning. Yet these methods provide either a stochastic behavior at a local level (mixed strategies, reinforcement learning), or they provide a global stochastic behavior at a global level (reinforcement learning with *online learning*) but fail to produce an optimal policy.

The presented method provides a global optimal strategy (for the depth of the computed game tree) given a level of entropy, while still simulating a stochastic behavior and following an optimal policy (for a degree of entropy θ) at the same complexity than simpler techniques (such as ϵ-greedy, or local

H	$\hat{\theta}$	$\mathbb{E}[C]_{Rminimax}$	$\hat{\epsilon}$	$\mathbb{E}[C]_{\epsilon-greedy}$
1.0	0.6349	1.8311	0.6679	1.9962
0.9	0.7496	1.6877	0.7187	1.8439
0.8	0.8692	1.5628	0.7665	1.7004
0.7	0.9937	1.4550	0.8080	1.5759
0.6	1.1280	1.3601	0.8456	1.4631
0.5	1.2745	1.2773	0.8798	1.3606
0.4	1.4454	1.2031	0.9106	1.2683
0.3	1.6505	1.1385	0.9384	1.1848
0.2	1.9239	1.0821	0.9628	1.1116
0.1	2.3634	1.0348	0.9838	1.0486

Table 4.3: Comparison of $\mathcal{R}$minimax and ϵ-greedy algorithms on the tree from Figure 4.12 for different levels of entropy H. Parameter estimations ($\hat{\theta}$ and $\hat{\epsilon}$) as well as the expected costs are reported.

Boltzmann exploration). Furthermore, the computation time is equivalent to the one of the MINIMAX algorithm.

The main drawback of this method is that paths are assumed to be uncorrelated and the opponent is assumed to be fully rational, both of which are not realistic for some problems.

Simulation experiments have led to the conclusion that the $\mathcal{R}$minimax algorithm behaves as expected. The compound of the $\mathcal{R}$minimax with pruning techniques, as well as techniques for reducing the search space, has demonstrated to be effective.

Future work will focus on four main areas: (i) investigating the extension of the $\mathcal{R}$minimax to multi-player games as well as online or dynamic games, (ii) to the estimation of a real player's θ parameter in order to mimic users' behavior and follow a similar learning curve, (iii) applying this framework to nested Monte-Carlo search techniques, and (iv) applying the RSP framework to Markov decision problems.

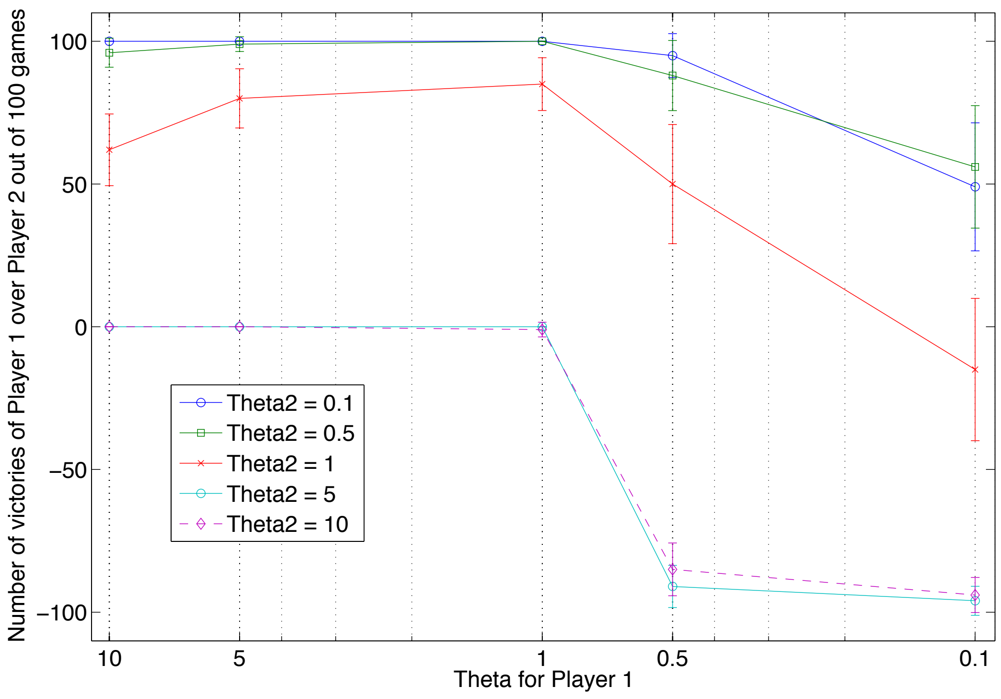


Figure 4.8: Resulting curves for the $\mathcal{R}$minimax algorithm for Tic-Tac-Toe. The algorithm is applied to the full game tree generated by the MINIMAX algorithm. The horizontal axis represents the variation of θ_1 for player π_1 while the vertical axis shows the number of victories of π_1 over π_2, out of 100 games. Each curve corresponds to a value of θ_2 for player π_2 with its 95% confidence intervals. The significant distance between curves is caused by the relative advantage of π_1 which always moves first. In a game with a limited space state like Tic-Tac-Toe, this difference is more marked than in games with broader space states, such as Connect-4.

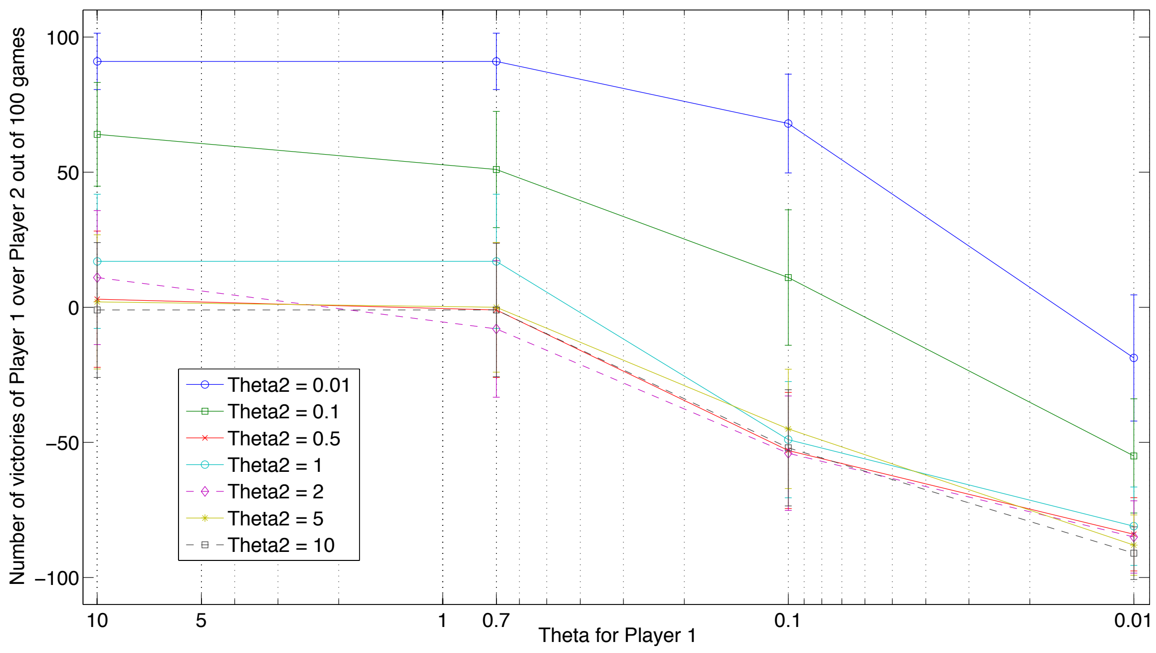


Figure 4.9: Resulting curves for the $\mathcal{R}$minimax algorithm applied to the game Connect-4. The algorithm is applied to a game tree of depth 5 generated by the MINIMAX algorithm, combined with heuristics. The horizontal axis represents the variation of θ_1 for player π_1 while the vertical axis shows the number of victories of π_1 over π_2, out of 100 games. Each curve corresponds to a value of θ_2 for player π_2.

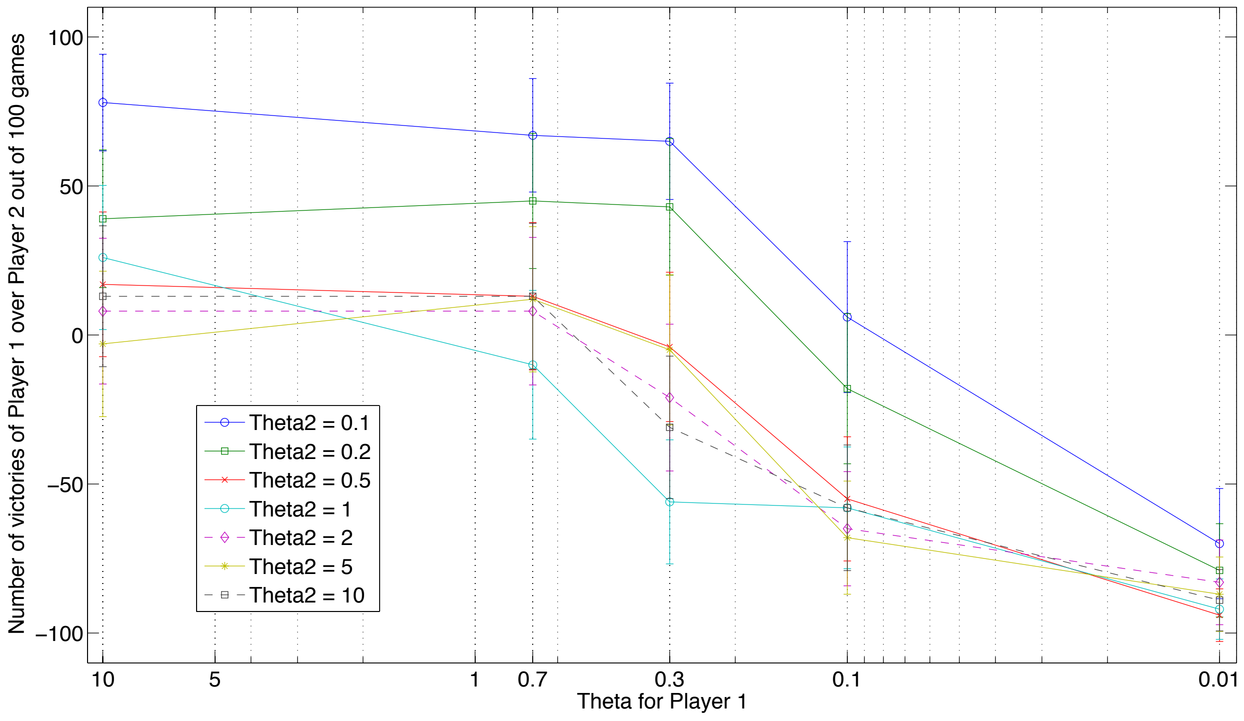


Figure 4.10: Resulting curves for the $\mathcal{R}$minimax algorithm applied to the game Connect-4. The algorithm is applied to a game tree of depth 5 generated by the alpha-beta algorithm combined with heuristics. The horizontal axis represents the variation of θ_1 for player π_1 while the vertical axis shows the number of victories of π_1 over π_2, out of 100 games. Each curve corresponds to a value of θ_2 for player π_2.

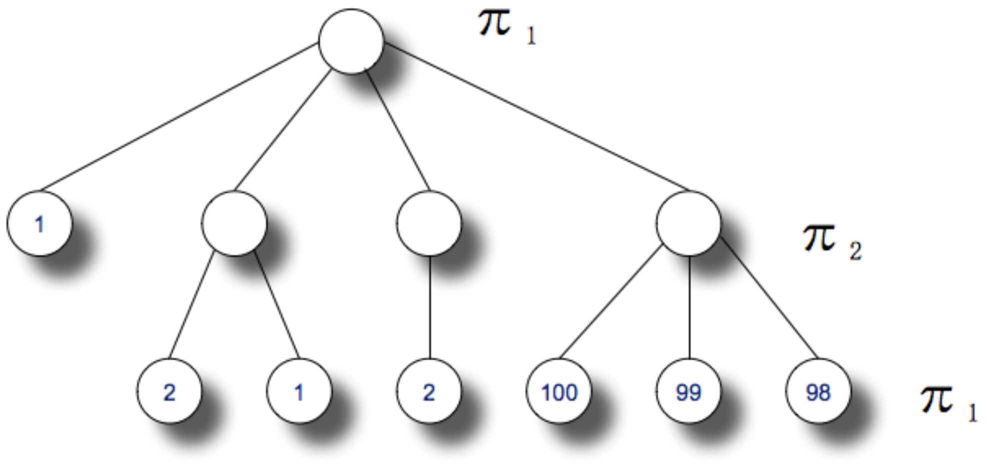


Figure 4.11: Game tree of depth 2 composed by eleven nodes for comparison between the $\mathcal{R}$minimax and the ϵ-greedy. Leaf nodes contain the utility values. π_1 plays MIN and π_2 plays MAX.

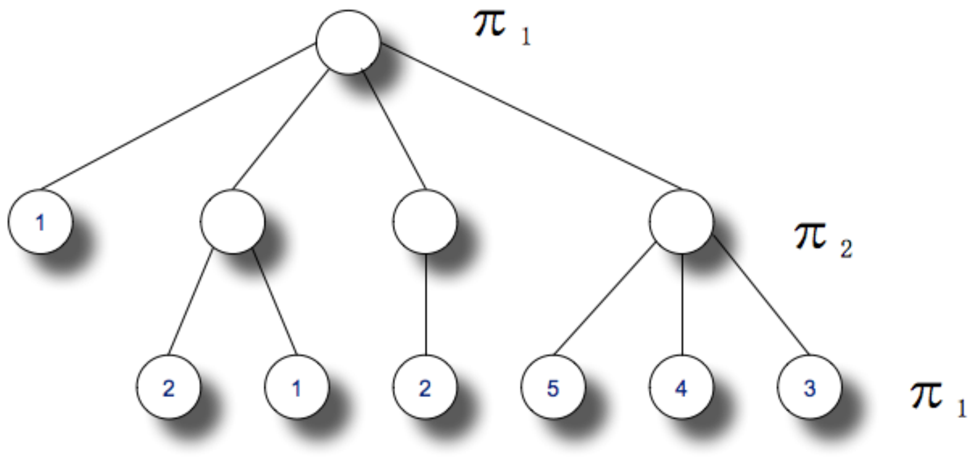


Figure 4.12: Game tree of depth 2 composed by eleven nodes for comparison between the $\mathcal{R}$minimax and the ϵ-greedy. Leaf nodes contain the utility values. π_1 plays MIN and π_2 plays MAX.

Chapter 5

A simple-cycles weighted kernel for music retrieval

Contents

The objective of this chapter is to exploit the cycles in a song's chord sequence. Indeed, we can consider the chord sequence of a song as a graph, where the natural repetitions in popular music are represented as cyclic motifs. These motifs are extracted and used in a novel Music Information Retrieval (MIR) system, which finds the songs whose harmonic sequence is closest to the queried song.

Let us take as example the sequence of chords from "Paradise" of Coldplay, as shown in Figure 5.1. The first thing we notice, is that there is a high level of repetition of chords, e.g., "Gm, F, Dm, C". If we build the graph made of all chord transitions, we will obtain the graph from Figure 5.2, where the sequence "Gm, F, Dm, C" is marked in orange, and the other two frequent chord sequences are marked in green and pink.

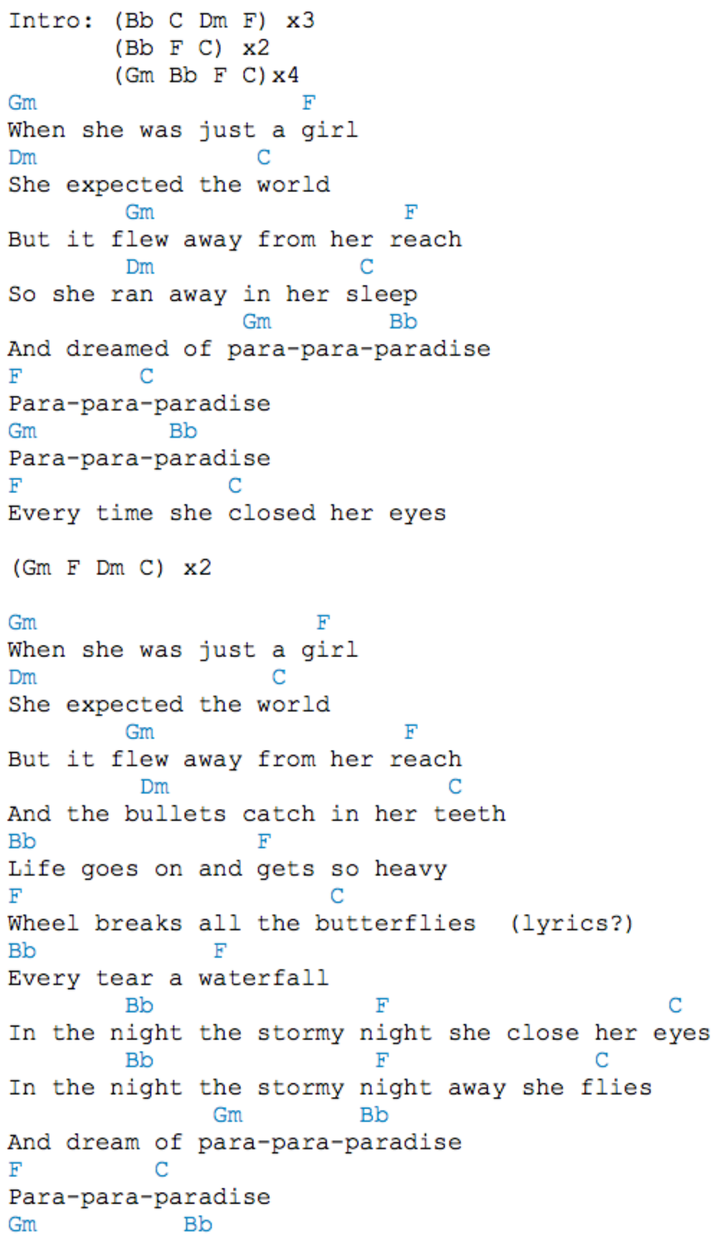
The first advantage of representing the chord sequence as a graph, is to offer a much more simplified and compressed representation of the song.

When transcribing a song in chord sequences, many people provide a partial representation, by omitting repetitions of the same sequence. The second advantage of using graphs, is to provide a common representation for versions of the same song, even if some portions are missing (as long as the main repeated sequences are presented once).

Furthermore, we know that many songs share similar chord subsequences, and a number of methods for extracting them is available. These repeated subsequences are also called simple cycles of a graph.

Motif extraction on graphs has attracted a lot of attention in the past years, e.g.,in community detection (Arenas et al., 2008), or in graph comparison (Horváth et al., 2004). A *motif* is formally defined in Arenas et al. (2008) as a connected undirected sub-graph (or weakly connected directed sub-graph) which appears frequently in a graph showing some kind of structure. Examples of motifs are cliques, paths, cycles, or sub-trees. The method presented in this chapter relies on the concept of cycle as a motif for similarity detection between graphs (isomorphism). By transforming the chord sequences into graphs, and comparing their simple cycles, we obtain a similarity measure based on the musical motifs of a song (see Section 5.6.1 for a more precise description).

The fourth advantage comes from music perception: since the beginning of the 15th century, motivic elements have made part of Western music, becoming common practice during the 18th century. We can find numerous examples of this phenomenon nowadays in modern pop/rock music which contain repetitive sub-structures, e.g., the chorus, verse, etc. According to Deliège et al. (1996),



```
Intro: (Bb C Dm F) x3
       (Bb F C) x2
       (Gm Bb F C)x4
Gm                  F
When she was just a girl
Dm               C
She expected the world
        Gm                 F
But it flew away from her reach
        Dm              C
So she ran away in her sleep
                Gm        Bb
And dreamed of para-para-paradise
F        C
Para-para-paradise
Gm         Bb
Para-para-paradise
F              C
Every time she closed her eyes

(Gm F Dm C) x2

Gm                   F
When she was just a girl
Dm                C
She expected the world
        Gm                 F
But it flew away from her reach
         Dm                    C
And the bullets catch in her teeth
Bb               F
Life goes on and gets so heavy
F                     C
Wheel breaks all the butterflies  (lyrics?)
Bb            F
Every tear a waterfall
        Bb               F                   C
In the night the stormy night she close her eyes
        Bb               F              C
In the night the stormy night away she flies
              Gm        Bb
And dream of para-para-paradise
F         C
Para-para-paradise
Gm          Bb
```

Figure 5.1: Extract of chord sequence of "Paradise" from Coldplay (`http://tabs.ultimate-guitar.com/c/coldplay/paradise_crd.htm`).

such repetitive structures, or *motifs*, act as *cues* in music perception. "A *cue* is a very restricted entity ... often shorter than the group itself, but always embodying striking attributes". This notion of cue, would let a listener encode

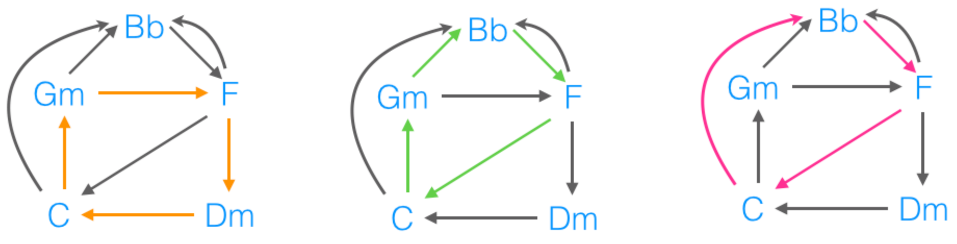


Figure 5.2: Simple cycles extracted from "Paradise" of Coldplay (`http://tabs.ultimate-guitar.com/c/coldplay/paradise_crd.htm`).

information in a more efficient way, allowing longer structures to be memorized by means of smaller, more salient, features.

Although motifs can be found in a song's harmony or melody, in this Chapter we focus on harmonic motifs for three reasons:

1. many songs share a part of their harmonic structure, as the number of chord progressions that are popular in a musical style (*idioms*) remain limited, while the melodic structure can vary greatly from one song to another;

2. studies in experimental psychology have shown the essential role of harmony in musical sequence perception (Drake, 1998);

3. although the amount of chord progression data is increasing thanks to chord estimation algorithms (see e.g.,Papadopoulos and Peeters (2011)) and user-generated data (which is readily available from the web), few efforts have been put on harmony-based similarity measures.

On the other hand, human listeners, due to their musical background, are more susceptible to like songs with a familiar harmonic structure, but yet different enough from the songs they already know (Paulus et al., 2010)[1]. We believe, thus, that comparing songs thanks to their harmonic motifs would

[1]This is explained by Pachet (1999) as 'the compromise between the repetition and the surprise" in the expectation of a human listener.

yield in a similarity measure that takes into account its repetitive harmonic sub-structures.

The technique presented in this chapter, unlike the previous ones, is not based on the RSP framework. Although Approximate String Matching techniques, like the SoP distances from Chapter 2, have already been extensively applied in melody comparison, we will not apply them in this chapter. The reason behind this decision is that harmonic sequences have a slightly different rationale than melodic ones. This makes sense, as melodic sentences can be repeated, but slight variations are often introduced.

On the other hand, harmony is articulated into short fundamental sentences which a high degree of repetition. This is the case, at least, of most of the commercial pop and rock music studied here (please note that more complex songs like the ones found in other styles like jazz, deserve further study as they may present much more complex harmonic structure).

Keeping the structure of harmonic sequences in mind, it seemed more natural to try to gather these repeated short sequences, which rarely vary, and use them for comparison. The technique presented in this chapter allows efficiently retrieving these short sequences, by performing an initial transformation from the harmonic sequence into a graph structure which serves as basis for the extraction.

The **research question** of this chapter is to investigate how to build a robust Music Information Retrieval system based on chord progressions and its relative performance regarding other string distances.

Main contributions

The contributions of the method presented in this chapter are as follows:

1. It is based on the **repetitive harmonic features** of songs (which can be easily extracted from web resources, as done in the present work).
2. The similarity measure deals with **large structural changes** in chord progression (e.g., addition of repetitions, bridge, etc.).
3. The similarity matrix can be extracted by means of **kernel functions**.

4. The similarity is **transposition invariant** (the intervals between chords are used, instead of the chords themselves).

5. We provide a simple, general, **methodology for computing similarities** from chord progressions (from the text mining step to acquire the data, to the automatic classification step with an SVM).

6. We exploit a **novel source of user-generated data** that is readily available on the Internet (in form of guitar chord progressions).

7. Empirical tests show that **music similarity retrieval** can be performed solely on the basis of chords.

5.1 Kernels in data mining and machine learning

In this section we introduce the kernel methods, which are the basis for the technique we present on this chapter. We show how to apply a kernel based on cyclic motifs on a graph as a means to compute a similarity measure between two graphs. The advantage of kernel methods is that they are efficient, robust, and statistically stable. The foundation of these methods is thoroughly presented in the work of Gärtner (2009); Shawe-Taylor and Cristianini (2004); Schölkopf and Smola (2002).

5.1.1 Basic notions of kernels

As explained by Shawe-Taylor and Cristianini (2004), the main idea behind kernel methods is that the observations we use as input data may not be linearly separable in their original *input space*. To overcome this issue, the data is projected into a new *feature space*, normally of higher dimension than the input space, where a linear separator, or hyper-plane, could be applied.

Let us illustrate this idea with a simple example: let us suppose we need to separate the data from Figure 5.3 in a two-dimensional space (the input space);

it is obvious from its representation that there is no linear separator in the 2-dimensional space that can solve this problem, i.e., separating observations from class A (in pink) and class B (in blue). Let us now map these observations into a higher dimensional space, the *feature space*, where there is an obvious hyper-plane that can linearly separate both classes (see Figure 5.4). This shows that increasing the dimensionality of the feature space could be a good way of achieving linear separability. Furthermore, thanks to the extensive research on the field of detecting linear relations, we have algorithms that are both efficient and well understood.

There is, however, a shortcoming of working with this new feature space: working with higher dimensional representations of linear patterns can be computationally expensive. Moreover, the mapping from the input space to the feature space could be difficult to compute in closed form. For this reason there is a computational shortcut that is known as the *kernel trick*. This kernel trick is a way of mapping the original observations into a higher dimensional space (inner product space) without computing this mapping explicitly, by means of a *kernel function*.

Shawe-Taylor and Cristianini (2004) summarize all kernel methods to the following three components, although their purpose and individual components may vary from one application to another:

1. A *mapping ϕ of the data* from the input space, x, into some meaningful, application-dependent, feature space, $\mathcal{F}$. The feature space allow us to find linear relations of the input data:

$$\phi : \mathbf{x} \to \phi(\mathbf{x}) \in \mathcal{F} \tag{5.1}$$

2. An *inner product* defined in the feature space, $\phi(\mathbf{x})$, so that there is no need for computing the coordinates of embedded points. The matrix containing the inner product of all pairs of observation is the kernel matrix, $\mathbf{K}$, (a positive semidefinite matrix of similarities). We assume that the kernel matrix can be computed efficiently thanks to the kernel

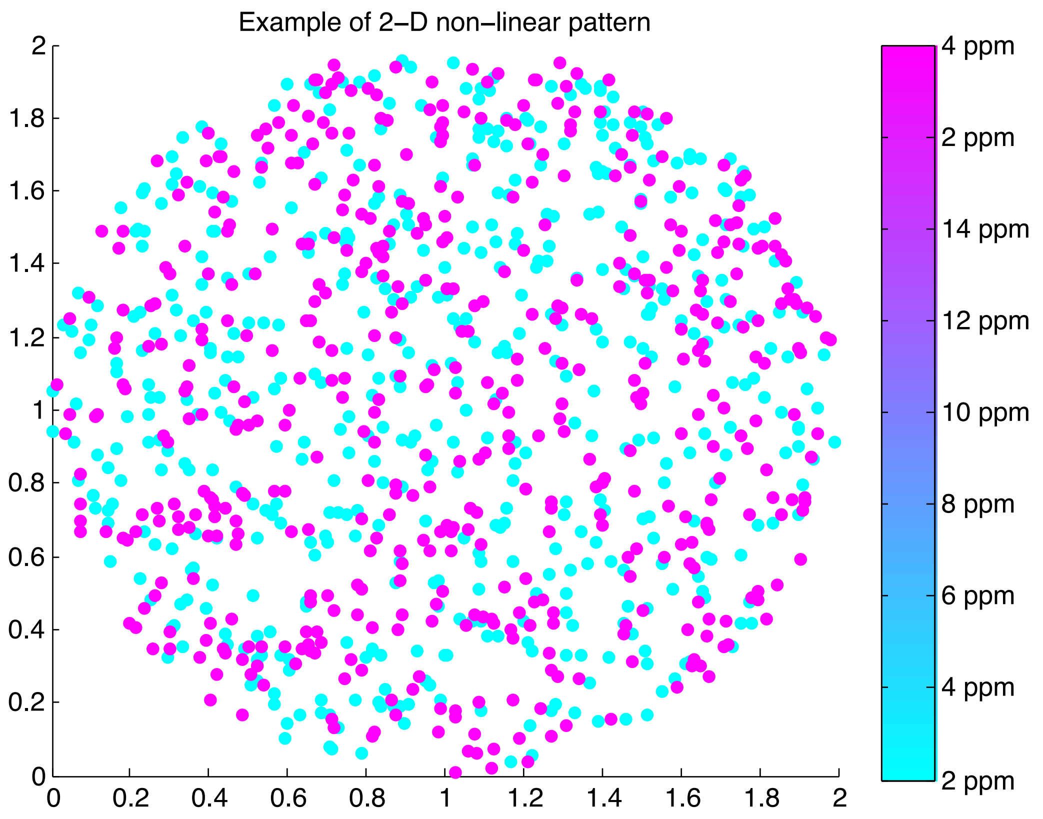


Figure 5.3: Example of a non-linear pattern in a 2-dimensional space. There are two classes of observations marked in pink and blue.

function:

$$k(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle \tag{5.2}$$

3. A *learning algorithm* for discovering patterns in that space, e.g., support vector machines, kernel principal component analysis, kernel k-means, etc. All these methods are based on the kernel matrix.

One interesting property of kernel functions is that, although the feature space may have infinite dimension (the number of possible cycles, in our case), it is often possible to compute them in polynomial time thanks to the kernel trick.

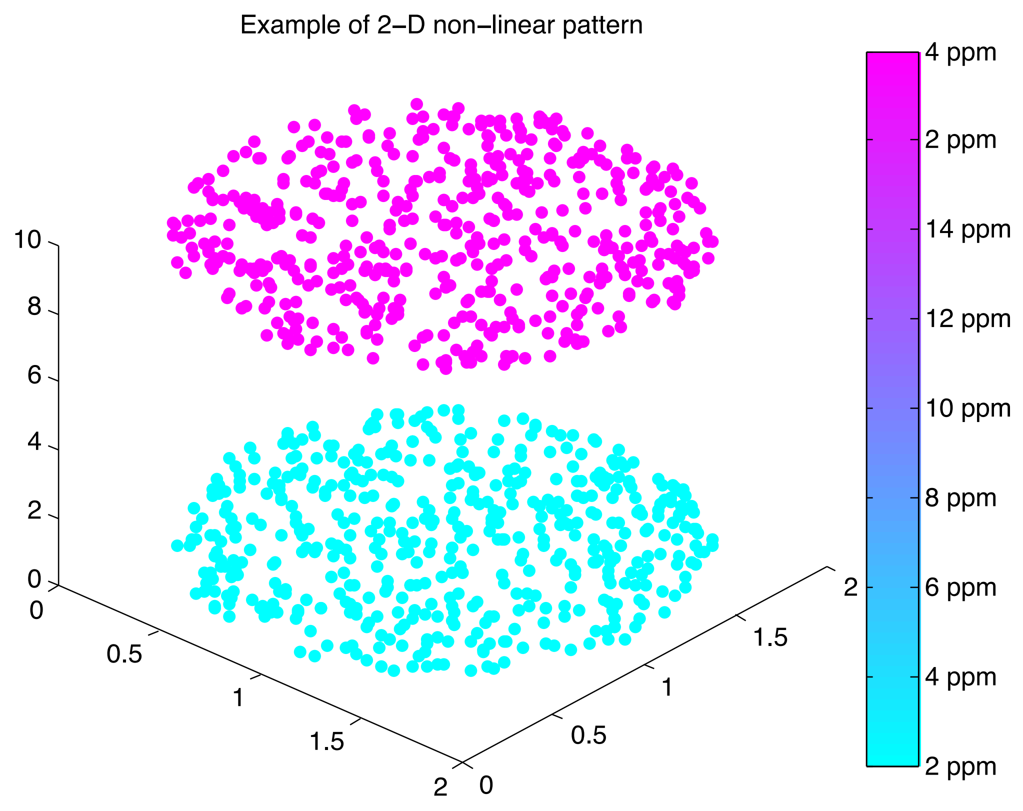


Figure 5.4: The same non-linear pattern in the 2D space projected in a 3-dimensional space. This example illustrates how the projection in a higher dimensional space can help the finding of a linear separator. In this case it would be the hyper-plane parallel to both sets of observations.

5.1.2 The Gram matrix

Shawe-Taylor and Cristianini (2004) present the example of linear regression in an input space to illustrate the concept of *Gram matrix* and *primal* and *dual solutions*. In linear regression we would like to find the real-valued linear function that best approximates a set of training points:

$$g(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle = \sum_{i=1}^{n} w_i x_i \tag{5.3}$$

where the training set contains l pairs of observations $\mathbf{x}_i \in \mathbb{R}^n$ and their output $y_i \in \mathbb{R}$, $S = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_l, y_l)\}$ (where $\mathbf{x}_i$ is a column vector). The form of linear regression presented in Equation (5.3) is known as the *primal solution*, where the weight vector, $\mathbf{w}$, is explicitly computed. Let us now assume that the weights are presented as a linear combination of the input vectors, where $\mathbf{w} = \sum_{i=1}^{l} \alpha_i \mathbf{x}_i = \mathbf{X}^T \boldsymbol{\alpha}$. In this case, the regression equation becomes

$$g(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle = \sum_{i=1}^{l} \alpha_i \langle \mathbf{x}_i, \mathbf{x} \rangle \tag{5.4}$$

Equation (5.4) corresponds to the *dual solution* for regression, where $\boldsymbol{\alpha}$ contains the *dual variables*. The main difference between both solutions, is that in the dual solution the information from the input data is given by their *inner products*. The matrix containing these inner products is called *Gram matrix* or *Kernel matrix*, $\mathbf{K} = \mathbf{X}\mathbf{X}^T$, which contains the evaluation of the kernel function on all pairs of data points.

It must be noted that in order to solve the dual equation, we need to solve a system of size l, as the dimension of $\mathbf{K}$ is $l \times l$. This may be more efficient than solving the system from Equation (5.3), as long as l, the number of training examples is higher than n, the dimension of the space. The drawback of the dual solution is that in order to evaluate the predictive function, the cost is $O(nl)$ compared to the $O(n)$ cost of the primal function.

5.1.3 Properties of kernels

Apart from the above-mentioned properties of efficiency, and statistical stability, kernel methods enjoy some other properties presented in this section, ranging from the mathematical basis to the construction of new kernels (Shawe-Taylor and Cristianini (2004)).

Kernel matrices are positive semidefinite. In order for a matrix to be a kernel it has to fill the condition of being positive semidefinite since it is defined as an inner product, or Gram matrix. In other words, all its eigenvalues are non-negative. Formally, a matrix $\mathbf{A}$ is positive semidefinite if and only if

$$\mathbf{v}'\mathbf{A}\mathbf{v} \geq 0 \; \forall \mathbf{v} \tag{5.5}$$

This is the case for kernels as stated in the following proof from Shawe-Taylor and Cristianini (2004) using the Gram matrix:

$$\begin{aligned}
\mathbf{v}'\mathbf{K}\mathbf{v} &= \sum_{ij=1}^{l} v_i v_j \mathbf{K}_{ij} && (5.6)\\
&= \sum_{ij=1}^{l} v_i v_j \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle && (5.7)\\
&= \left\langle \sum_{i=1}^{l} v_i \phi(\mathbf{x}_i), \sum_{j=1}^{l} v_j \phi(\mathbf{x}_j) \right\rangle && (5.8)\\
&= \left\| \sum_{i=1}^{l} v_i \phi(\mathbf{x}_i) \right\|^2 \geq 0 && (5.9)\\
& && (5.10)
\end{aligned}$$

Now that we know the intution of what is a kernel, we can create new **complex kernels by combining with basic operations simpler kernels**. Let us suppose that we want to improve an existing kernel, or add extra information by combining two different kernels. For achieving this, we define the operations that can be applied to kernels, without loss of the positive semidefinite property. In other words, we can consider that the class of kernel functions is closed under the following operations (Shawe-Taylor and Cristianini (2004)):

1. $k(\mathbf{x}, \mathbf{z}) = k_1(\mathbf{x}, \mathbf{z}) + k_2(\mathbf{x}, \mathbf{z})$ where k_1 and k_2 are kernel functions over $\mathbb{R}^n$
2. $k(\mathbf{x}, \mathbf{z}) = ak_1(\mathbf{x}, \mathbf{z})$ where a is a real value
3. $k(\mathbf{x}, \mathbf{z}) = k_1(\mathbf{x}, \mathbf{z})k_2(\mathbf{x}, \mathbf{z})$
4. $k(\mathbf{x}, \mathbf{z}) = f(\mathbf{x})f(\mathbf{z})$ where $f(\cdot)$ is a real-valued function
5. $k(\mathbf{x}, \mathbf{z}) = k_3(\phi(\mathbf{x}), \phi(\mathbf{z}))$ where k_3 is a kernel function over $\mathbb{R}^m$

6. $k(\mathbf{x}, \mathbf{z}) = \mathbf{x}'\mathbf{B}\mathbf{z}$ where $\mathbf{B}$ is a symmetric positive semidefinite $n \times n$ matrix

Chapter 5 shows how to produce a normalized weighted kernel for simple cycles by using some of these basic operations.

5.2 Information Retrieval

This section is an introduction to the basic concepts of *Information Retrieval* (IR) systems (Baeza-Yates and Ribeiro-Neto, 2011; Manning et al., 2008). We have applied these principles to build a prototype IR system with the data presented in this chapter. Although the IR system presented in this work is of much smaller scale, the same principles can be applied.

5.2.1 Basic notions of Information Retrieval

As described by Baeza-Yates and Ribeiro-Neto (2011), an information retrieval model is a quadruple $[\mathcal{D}, \mathcal{Q}, F, R(q_i, d_j)]$ where:

- $\mathcal{D}$ is a set of representations of documents in a collection. Please note that a document in this context is any piece of information that we want to retrieve. In our case, a document is the chord sequence of a song.
- $\mathcal{Q}$ is a set of representations of user queries. The queries represent the information that users wish to retrieve. In our case, the query retrieves all similar songs to a given one.
- F is a framework that models both document representations, and query representations. Furthermore, it establishes the relationship between both, e.g., vectors, linear algebra operations, probability distributions, etc.
- $R(q_i, d_j)$ represents the ranking function that assigns a value to each resulting document $d_j \in \mathcal{D}$ when performing query $q_i \in \mathcal{Q}$. This value is used for sorting the result set of documents by relevance.

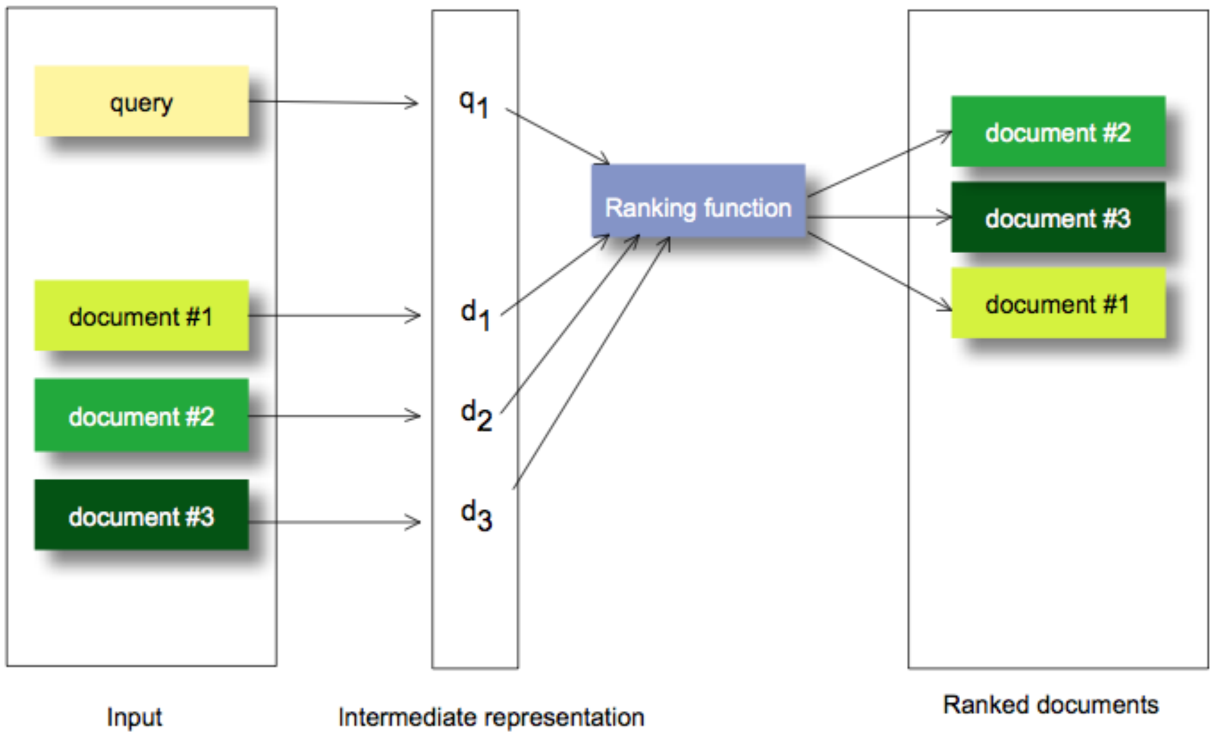


Figure 5.5: Information Retrieval process example with one query and three documents.

As shown, the main purpose of such systems is to retrieve the most relevant collection of documents to a given user query. Figure 5.5 summarizes the steps of an Information Retrieval system. We briefly explain in the next section the different kinds of IR systems, their advantages, and drawbacks.

5.2.2 Types of IR systems

As mentioned by Moens (2006) we can classify IR systems depending on the way users search for information, i.e., (i) key term search, (ii) question answering, and (iii) query by example. *Key term search* is a very popular paradigm of querying, where the user provides some input keywords which are matched against the document collection. These systems can be further extended by adding word disambiguation (e.g., “windows” can both refer to the window in a building, or to the OS Windows) and concept search (a more refined method which takes into account the semantics of the queried keywords and phrases).

Question answering, on the other hand, tries to provide an answer to a natural language question, e.g.,“What is the procedure to obtain a resident card in Belgium?”. The objective here is to find this information from the document collection, or at least to extract the paragraphs that are useful and rank them by relevance. These complex systems are based on Information Extraction (IE) techniques and reasoning.

A different search paradigm is given by the *query by example*, very popular in multimedia IR systems, where users cannot describe accurately the query they want to make. Instead, they will provide an example, e.g., a musical piece or an image, and let the system retrieve the most similar cases.

Another approach is presented by Baeza-Yates and Ribeiro-Neto (2011), who classify IR systems according to the type of document they try to retrieve, i.e., multimedia, web sites, and text documents. *Multimedia retrieval*, as mentioned before, focuses on searching images, audio and video over a corpus of documents. These retrieval techniques comprise a series of steps to transform the signal into an intermediate representation which can be easily queried. In the case of *web retrieval*, text only is not enough to be able to classify a website. For this reason, the use of link information between documents is frequently used as part of the model. The most well-known models are Page-Rank from Page et al. (1999), Hubs & Authorities by Ding and He (2004), and the work from Kleinberg (1999).

On the other hand, *text document retrieval* searches documents whose text matches a given query, which is usually encoded as a set of keywords, a sentence, or a combination of both. This subject has been the objective of extensive research, and it is the field where we find the largest number of IR models. These models can refer to either semi-structured text or unstructured text. In the first category, specific parts of the document such as the title and sections are used. These parts contain unstructured text, but they are organized partially. These models include the XML-based indexing methods, and proximal nodes (see, e.g., Baeza-Yates and Ribeiro-Neto (2011) for more details).

In the second category of text-based models, text is represented as a sequence of words, a series of keywords, or a combination of both. There are three basic types of models for *unstructured text retrieval*:

1. The Boolean model, where documents and queries are expressed as indexed terms or words. These models are based on set theory.

2. The Probabilistic model, where documents and queries are represented by probability distributions. We say that these models are probabilistic.

3. The Vector model, where documents and queries are represented by vectors in a high dimensional space. These models are algebraic.

Let us now present further these three models, their main characteristics, and advantages.

5.2.3 The Boolean model

The *Boolean model* (Baeza-Yates and Ribeiro-Neto, 2011) is based on set theory and Boolean algebra. This model is very simple, intuitive, but also very limited. Let us illustrate this model with an example, but first some concepts need to be defined:

- Index terms: these are the words that appear in a document, although it can also be a group of words. The set of all terms is our vocabulary $V = \{k_1, k_2, ..., k_t\}$.

- Document representation: once our vocabulary is defined, we can define a document or a query as a vector with 1 in the position of a contained term, and 0 in the remaining positions. E.g., $\mathbf{d}_i = [0, 1, 1, 0]$ means that only the second and third terms of our vocabulary of size four occur in the document i.

- The term-document matrix: this matrix contains one row per word in the vocabulary, and one column per document in the collection. Its entries f_{ij} represents the frequency of term k_i in document $\mathbf{d}_j$.

The above definitions apply to all models. However, there are other elements which are specific to the Boolean model. In this case, the query representation is a Boolean expression on index terms, e.g., q $= [k_3 \wedge (k_1 \vee k_2)]$. Furthermore, the similarity function is defined as

$$sim(\mathbf{d}_j, \mathrm{q}) = \begin{cases} 1 & \text{if } \mathbf{d}_j \text{ satisfies query q} \\ 0 & \text{otherwise} \end{cases} \tag{5.11}$$

Table 5.1 presents an example of the Boolean model with its main elements.

Vocabulary	$V = \{\mathrm{cat}, \mathrm{dog}, \mathrm{animal}\}$.
Documents	$\mathbf{d}_1 = [1,0,1], \mathbf{d}_2 = [0,1,1], \mathbf{d}_3 = [0,0,0], \mathbf{d}_4 = [1,1,1]$
Query	$\mathrm{q} = [k_3 \wedge (k_1 \vee k_2)] = [\mathrm{animal} \wedge (\mathrm{cat} \vee \mathrm{dog})]$
Term-document matrix	$\begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \end{pmatrix}$
Similarities	$sim(\mathbf{d}_1, \mathrm{q}) = 1$, $sim(\mathbf{d}_2, \mathrm{q}) = 1$, $sim(\mathbf{d}_3, \mathrm{q}) = 0$, $sim(\mathbf{d}_4, \mathrm{q}) = 1$

Table 5.1: Boolean model example with a vocabulary of size 3, a collection of 4 documents, and a query. The query tries to identify the documents which contain term k_3 and at least one of the k_1 or k_2 terms.

The reader may remark that there is no notion of partial match in this similarity function. Indeed, in the Boolean model, a document can be either relevant or non-relevant, being no intermediate classification in between. This way of ranking documents is quite poor, as it retrieves a very large number of documents or too few of them. For this reason, term weighting was introduced by Jones (1972) and Salton and Yang (1973) as a clear improvement to this basic model, as explained below.

We associate to each term k_i in a document $\mathbf{d_j}$ a weighting element, w_{ij}, that allows defining the relevance of the term. Words appearing in a restricted number of documents are more meaningful, and therefore more relevant, than words appearing in every document; the first having a higher weight than the latter.

Other main improvements to this model are:

- The *term-term correlations*: it may happen that the appearance of a term in a document increases the probability of appearance of related terms, e.g., in a set of documents about computing, the terms “computer” and “science” should be correlated and appear together. The association of two terms k_u and k_v is computed as

$$c_{uv} = \sum_{\mathbf{d_j} \in \mathcal{D}} w_{uj} w_{vj} \tag{5.12}$$

  Although this improvement increases the relevance of the obtained ranking, many IR systems prefer to simplify their computational cost by assuming term independence.

- The *TF-IDF weighting*: this weighting method is the most popular among IR systems and was introduced by Salton and Yang (1973). It combines two elements: (i) the frequency of a term in the collection of documents (TF), and (ii) the inverse document frequency (IDF). The first element measures the frequency of a term in a document; thus, the more frequent is a term in a document, the more relevant. The second element measures the “rareness” of the term in the whole document collection; thus, the rarer the term in the collection, the more relevant. The TF-IDF for a term k_i and a document $\mathbf{d}_j$ can be computed as:

$$w_{ij} = \begin{cases} (1 + \log f_{ij}) \log \dfrac{N}{n_i} & \text{if } f_{ij} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.13}$$

  where N is the number of documents in the collection, and n_i is the number of documents where k_i appears. This method works fine for general collections, where no other information about term weights is available.

- The *document length normalization*: because longer documents contain more terms, they are also more likely to be retrieved by an IR system. To avoid favoring long documents, some normalization is usually applied (the document length is divided by its norm). The most common approach is to take the vector norm which considers a document to be a vector of weighted terms:

$$|\mathbf{d}_j| = \sqrt{\sum_{i=1}^{t} w_{ij}^2} \tag{5.14}$$

  where t is the total number of terms.

5.2.4 The Probabilistic model

The *Probabilistic model* (Manning et al., 2008) assumes that, given a certain query, there exists a set of documents which are relevant to the query. It also assumes that there is an underlying probability model based on the terms, that could generate that set of documents. As the exact set of documents is unknown, the probabilistic model estimates the probability distribution of those terms based on an initial set of documents. The system includes all feedback from users to improve its model, which becomes more and more accurate.

This model, which was introduced by Robertson (1977), states that given a query $\mathbf{q}$ and a document $\mathbf{d}$, the probabilistic model estimates to which extent a user will find the document relevant to the query. The similarity of a document is defined as:

$$sim(\mathbf{d}, \mathbf{q}) = \frac{\mathrm{P}(\mathbf{d} \in R | \mathbf{d}, \mathbf{q})}{\mathrm{P}(\mathbf{d} \in \overline{R} | \mathbf{d}, \mathbf{q})} \tag{5.15}$$

where R represents our random variable of relevant documents, and $\overline{R}$ is its complement (the non-relevant ones). By applying Bayes' rule we obtain:

$$sim(\mathbf{d},\mathbf{q}) = \frac{\mathrm{P}(\mathbf{d} \in R,\mathbf{q})\mathrm{P}(\mathbf{d}|\mathbf{d} \in R,\mathbf{q})}{\mathrm{P}(\mathbf{d} \in \overline{R},\mathbf{q})\mathrm{P}(\mathbf{d}|\mathbf{d} \in \overline{R},\mathbf{q})} \tag{5.16}$$

$$= \frac{\mathrm{P}(\mathbf{d} \in R|\mathbf{q})\mathrm{P}(\mathbf{d}|\mathbf{d} \in R,\mathbf{q})}{\mathrm{P}(\mathbf{d} \in \overline{R}|\mathbf{q})\mathrm{P}(\mathbf{d}|\mathbf{d} \in \overline{R},\mathbf{q})} \tag{5.17}$$

Since $\mathrm{P}(\mathbf{d} \in R|\mathbf{q})$ and $\mathrm{P}(\mathbf{d} \in \overline{R}|\mathbf{q})$ are scaling factor common to the whole document collection, we obtain:

$$sim(\mathbf{d},\mathbf{q}) \sim \frac{\mathrm{P}(\mathbf{d}|\mathbf{d} \in R,\mathbf{q})}{\mathrm{P}(\mathbf{d}|\mathbf{d} \in \overline{R},\mathbf{q})} \tag{5.18}$$

We can now express these equations in function of the terms k_i in the documents, and after some calculations and assuming independence between terms (see, e.g., Baeza-Yates and Ribeiro-Neto (2011) for more details), we obtain:

$$sim(\mathbf{d}_j,\mathbf{q}) \sim \sum_{k_i \in q \wedge k_i \in \mathbf{d}_j} \log\left(\frac{\mathrm{P}(k_i|R,\mathbf{q})}{1-\mathrm{P}(k_i|R,\mathbf{q})}\right) + \log\left(\frac{1-\mathrm{P}(k_i|\overline{R},\mathbf{q})}{\mathrm{P}(k_i|\overline{R},\mathbf{q})}\right) \tag{5.19}$$

Please note that although we are taking the log of the previous similarity, this will not affect the ranking as the logarithm is a monotonic increasing function.

However, there is a main issue for this method, as we do not have R initially. In order to be able to compute the probabilities $\mathrm{P}(k_i|R,\mathbf{q})$ and $\mathrm{P}(k_i|\overline{R},\mathbf{q})$, let us first introduce the Term Incidence Contingency Table from Robertson and Jones (1976):

	Relevant	Non-relevant	Total
Documents that contain k_i	r_i	$n_i - r_i$	n_i
Documents that do not contain k_i	$R - r_i$	$N - n_i - (R - r_i)$	$N - n_i$
All documents	R	$N - R$	N

Table 5.2: Term Incidence Contingency Table.

By using the information from the Contingency Table we can express the probabilities as:

$$\mathrm{P}(k_i|R,\mathbf{q}) = \frac{r_i}{R} \tag{5.20}$$

$$\mathrm{P}(k_i|\overline{R},\mathbf{q}) = \frac{n_i - r_i}{N - R} \tag{5.21}$$

We can assume, for now, that we do not have any estimations for r_i, nor R. The similarity measure then yields (for $R = r_i = 0$):

$$sim(\mathbf{d}_j,\mathbf{q}) \sim \sum_{k_i \in q \wedge k_i \in \mathbf{d}_j} \log\left(\frac{N - n_i + 0.5}{n_i + 0.5}\right) \tag{5.22}$$

Let us illustrate this model with an example (see Table 5.3). It can be observed that $\mathbf{d}_2$ is ranked higher, as the term it contains appears in only one document, and is considered as more relevant than the common term in $\mathbf{d}_1$ and $\mathbf{d}_3$. This can be considered as an IDF component, although no term frequencies are taken into account.

Vocabulary	$V = \{\text{cat}, \text{dog}, \text{animal}\}.$
Documents	$\mathbf{d}_1 = [0,1,1], \mathbf{d}_2 = [1,0,0], \mathbf{d}_3 = [0,0,1]$
Query	$\mathbf{q} = [1,0,1]$
Term-document matrix	$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{pmatrix}$
Similarities	$sim(\mathbf{d}_1,\mathbf{q}) = \log\left(\frac{3-2+0.5}{2+0.5}\right) = -0.5108$ $sim(\mathbf{d}_2,\mathbf{q}) = \log\left(\frac{3-1+0.5}{1+0.5}\right) = 0.5108$ $sim(\mathbf{d}_3,\mathbf{q}) = \log\left(\frac{3-2+0.5}{2+0.5}\right) = -0.5108$

Table 5.3: Probabilistic model example with a vocabulary of size 3, a collection of 3 documents, and a query. The query tries to identify the documents which contain term k_1 and k_3.

This model assumes term independence, which is unrealistic, and it also lacks document length normalization nor term frequencies. Several extensions to this model exist to overcome the length normalization and term frequencies. It is, however, a method that ranks documents according to their optimal probability of being relevant to the user query.

5.2.5 The Vector Space model

The last category of models is the *Vector model*, introduced by Salton et al. (1975), and which takes advantage of the before mentioned improvements, i.e., term weighting, document length normalization, etc. It proposes a model with partial matching by considering a document as a vector, $\mathbf{d}_i$, of weighted terms in a t-dimensional space, and further defining the similarity between a document and a query as the cosine of the angle between them. Usually, the cosine similarity or inner product is used:

$$sim(\mathbf{d}_j, \mathbf{q}) = \frac{\mathbf{d}_j \cdot \mathbf{q}}{\|\mathbf{d}_j\| \|\mathbf{q}\|} = \frac{\sum_{i=1}^{t} w_{ij} w_{iq}}{\sqrt{\sum_{i=1}^{t} w_{ij}^2} \sqrt{\sum_{i=1}^{t} w_{iq}^2}} \tag{5.23}$$

By applying the vector norm, we already perform document length normalization. Obviously, all improvements presented in the previous section, e.g., the TF-IDF weighting scheme can be applied. As a last step, all documents are sorted by their similarity with the query, providing a ranking of the document collection.

Let us illustrate this model with an example. In Table 5.4 we use a simple weighting which corresponds to the term frequencies.

The main advantages of this model are the improved ranking thanks to the weighting of the terms, the possibility of partial matching, and the document length normalization, at the cost of assuming term independence. It is agreed that, for general collections, this is the best basic retrieval model. For this reason, we apply this model category in the IR system from Chapter 5. However, it makes the assumption that the terms on a document are uncorrelated, and the term vectors are pair-wise orthogonal.

Vocabulary	$V = \{\text{cat}, \text{dog}, \text{animal}\}$.
Documents	$\mathbf{d}_1 = [2,0,1], \mathbf{d}_2 = [0,2,1], \mathbf{d}_3 = [0,0,1], \mathbf{d}_4 = [3,1,1]$
Query	$\mathbf{q} = [2,1,3]$
Term-document matrix	$\begin{pmatrix} 2 & 0 & 0 & 3 \\ 0 & 2 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$
Similarities	$sim(\mathbf{d}_1, \mathbf{q}) = 0.8367$, $sim(\mathbf{d}_2, \mathbf{q}) = 0.5976$, $sim(\mathbf{d}_3, \mathbf{q}) = 0.8018$, $sim(\mathbf{d}_4, \mathbf{q}) = 0.8058$

Table 5.4: Vector model example with a vocabulary of size 3, a collection of 4 documents, and a query. The query tries to identify the documents which contain twice term k_1, once term k_2, and three times term k_3.

5.3 Basic notions on tonality

This Section presents a brief introduction to tonality and harmony definitions used in the remaining of the chapter for non-musician readers. Tonality is a highly structured musical system in which pitches or chords are arranged by a hierarchy of perceived relations, stabilities, and attractions.

A *pitch* in music refers to the frequency of a sound, determining whether a sound is perceived as high or low. Although the pitch or frequency may be equal in two songs, the perception of the pitch may differ, as studied by psychoacoustics. We define a *chord* as multiple harmonic pitches that sound simultaneously. The notion of harmonic pitches is defined by the differences between frequencies, i.e., some frequency intervals are "harmonic" while others are not. These intervals are quantified by the notion of *tones* or half-tones.

The pitch or chord with the greatest stability is called the tonic. If we think about the scale of C major, "C, D, E, F, G, A, B, C", we can observe that:

- Each scale contains 8 notes, where the first note is repeated at the end. A new scale can be constructed, e.g., by shifting one position.
- The first note is called **tonic**, as it is the note that resolves musical tensions. In our example, that would be "C". Another important note in

the scale would be the **dominant** which corresponds to the 5th note of the scale, i.e., “G”. We can build a triad chord by combining the tonic, the dominant and the **mediant** “E”: “C-E-G”.

- The distance that separates two notes, or **interval**, can be either one full tone, or a semi-tone. In a major scale, all notes are separated by an interval of one tone, except the intervals between the 3rd and 4th note, and the 7th and 8th note. If we take the major scale of “D”, we will need to introduce **alterations**, i.e., flat ♭ and sharp ♯ to decrease or increase respectively a semi-tone, so to preserve the intervals between notes: “D, E, F♯, G, A, B, C♯, D”.

- The musical **key** of a song is the **tonic** of the song. e.g., if we have a song where the most repeated chords are “C”, “G”, and “F”, where the remaining chords have no alteration, we can infer with high probability that the song is in the “C” major key. This process needs to compare with all existing scales, and identifying the most important notes.

- Two different notes can have the same sound if their interval is zero, it is what we call **enharmonics**, e.g., “B♯” and “C”. These two notes sound the same, as the interval between “B” and “C” is a semi-tone, and by increasing a semi-tone “B” we reach the “C” sound. Other example is “A♭♭” and “G”, where we decrease of one tone “A” until reaching the “G” sound.

- All major **scales are equivalent** among them (same applies to other types of scales), as the intervals between notes in the same position are equal. This means that the scale of “C major” sounds exactly as the scale of “D major”. Some composers prefer to use one or another scale for practical purposes when playing guitar (some chords are more difficult to play, or the sound is too low for a singing voice). However, it takes a trained ear to distinguish between major scales. A normal person will only hear the intervals of the song, i.e., how the tension is built, how the phrase is released, etc. This phenomenon can be compared to speed (when constant, even if it has different values, it is not perceived) and acceleration (which is what we perceive) in mechanical physics.

- We define the **transposition** of a song as shifting all chords of a song one or several tones up or down. This means that, two songs can sound equal, but be represented with a different set of chords. This will be taken into account when comparing chord sequences.

5.4 Related work

Harmonic similarity has recently attracted the attention of the MIR (Music Information Retrieval) community thanks to the improvement in chord estimation techniques (Papadopoulos and Peeters, 2011), as well as the increase of the available data. One of the advantages of harmonic similarity is its ability to infer similar songs whose melodies differ. In this context, de Haas et al. (2008) proposes an approach based on the Tonal Pitch Space (TPS) which compares the change of chordal distance to the tonic over time. This local distance is then used to build a step function that computes the global distance between two chord progressions by minimizing the non-overlapping area of the two step functions. However, this method requires information about the key of the piece and does not support structural changes (e.g., introduction of repetitions).

We can also find techniques based on approximate string matching, such as the one proposed by Hanna et al. (2009). This technique extracts the most similar regions of the two chord sequences, and computes a distance based on the number of simple operations (insertion, deletion, substitution) that are needed to transform the first region into the second. This algorithm has complexity $O(nm)$ where n and m are the length of the sequences to compare, and edition costs must be provided.

Generative models are the third type of harmony similarity techniques. Such models assume that harmony variations occur according to an underlying model. Pickens and Crawford (2002) propose to model chord transitions of a song by means of a nth-order Markovian model, which serves as basis for a Kullback-Leibler scoring function. A generative model based on linguistics has also been applied by de Haas et al. (2009). This harmony similarity method is based on the assumption of a generative grammar of tonal harmony. By applying a weighted version of this grammar, a unique parse tree representing the chord sequence is obtained for each song – note that context free grammars produce multiple ambiguous parse trees, thus a weighting of the rules is needed to choose among all possibilities. In order to measure the similarity of a pair of parse trees, the largest embeddable tree is extracted. However, its time complexity is $O(min(n, m)nm)$ and this technique may reject a sequence which is considered as ungrammatical.

5.5 Cyclic patterns kernel

Cyclic patterns represent harmonic motifs in chord progressions. In order to extract these motifs for music similarity, we rely upon the cyclic pattern kernels from Horváth et al. (2004). This section presents the key concepts of this technique which computes a kernel based on the set of cyclic and tree patterns of a labelled graph.

5.5.1 Graphs and cycles

Let us first give some definitions concerning graphs and cycles. Let $G = (V, E, label)$ be a directed labelled graph defined as a finite set of vertices V, edges $E \subseteq [V]^2$, and their labels. The cardinalities of V and E are n and m, respectively. We define a *simple cycle* on G as a sequence

$$C = \{v_0, v_1\}, \{v_1, v_2\}, ..., \{v_{k-1}, v_k\} \tag{5.24}$$

where $v_0 = v_k$ and all others $v_i \neq v_j$ for every i, j $(1 \leq i \leq j \leq k)$. Although a cycle may have several permutations, only one of them (and the same in all cases) will be kept for our purposes. We can now define the set $\mathcal{S}(G)$ as the set of simple cycles of G, the set of unrestricted cyclic patterns (which are not necessarily simple) as $\mathcal{C}(G)$, and its relation:

$$\mathcal{S}(G) \subset \mathcal{C}(G) \tag{5.25}$$

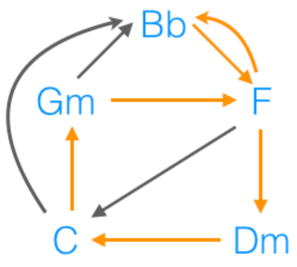


Figure 5.6: Non-simple cycle.

In the example from Figure 5.6 we can see that node “F” is repeated in the cycle marked in orange, and therefore it is not a simple cycle. The orange cycle, however, is composed of two simple cycles: F-Dm-C-Gm and F-Bb.

Similarly, we can define the set of *tree patterns*, $\mathcal{T}(G)$, as:

$$\mathcal{T}(G) = \{T \text{ is a connected component of } \mathcal{B}(G)\} \tag{5.26}$$

where $\mathcal{B}$ is the set of bridges of G (see Horváth et al. (2004) for more details). The method from Horváth et al. (2004) is based on these definitions, and finds iteratively the simple cycles of a graph by adding a connected sub-component at each step. These connected sub-components are connected thanks to the bridges.

5.5.2 Cyclic patterns kernel function

A cyclic patterns kernel function is proposed by Horváth et al. (2004), which takes two graphs as input, extracts their cyclic $\mathcal{C}(G)$ and tree patterns $\mathcal{T}(G)$ and uses them to build a mapping $\Phi_{\text{CP}}(G)$ into the feature space:

$$\Phi_{\text{CP}}(G) = f(\mathcal{C}(G) \cup \mathcal{T}(G)) \tag{5.27}$$

The *cyclic pattern kernel* is defined as the set of all simple cycles and tree patterns that appear in both graphs:

$$k_{\mathrm{CP}}(G_i, G_j) = |\mathcal{C}(G_i) \cap \mathcal{C}(G_j)| + |\mathcal{T}(G_i) \cap \mathcal{T}(G_j)| \qquad (5.28)$$

However, the problem of computing cyclic pattern kernels is NP-hard. For overcoming this issue, Horváth et al. (2004) restrict the set of cyclic patterns to $\mathcal{S}(G)$, so that only *simple cycles* are computed (those cycles whose only repeated nodes are the first and the last one). The advantage of simple cycles is that they can be computed in polynomial time. The authors use the algorithm from Read and Tarjan (1975), which extracts the simple cycles of a graph by means of a depth-first search in time $O(n+m(c+1))$, where n is the number of vertices, m is the number of edges, and c is the number of simple cycles. It is important, thus, that there exists a bound (*well-behaved* data) on the number of simple cycles for the sake of efficiency of the algorithm. As empirically shown in Section 5.8.2 (see Figure 5.11), this is the case for our chord data. Please note that, although the method from Horváth et al. (2004) computes simple cycles and trees, we will use the simple cycles for our purpose.

Although this method relies on the technique proposed by Read and Tarjan (1975) for finding these cycles, the method we have chosen because of simpler implementation is the one of Ponstein (1966). This method computes the self-avoiding paths based on the adjacency matrix of the graph, which are equivalent to the simple cycles. The method of Ponstein (1966) is less efficient than the one of Read and Tarjan (1975), but it has a simpler implementation.

5.6 Simple-cycles weighted kernel

In this section we present the proposed kernel, which is an extension of the cyclic pattern kernels from Horváth et al. (2004) introduced in the previous section. We propose to focus our kernel only on simple cycles which will represent the repetitive harmonic sub-structure of a song. In order to favor longer simple cycles, a weighted (normalized) version of the kernel will be computed.

5.6.1 Graph extraction

Chord sequences represent the harmonic progression of a song which may modulate over time, i.e., its key changes through time. This is an important issue for the detection of harmonic similarities, as the transposed chords may not coincide. In order to make our method transposition-invariant we will thus convert the chord sequence into interval sequences, from which input graphs will be extracted. As only structure matters for us, and not the "musical distance" between a pair of chords (in semi-tones), a label λ_i will be assigned to each chord transition with the same "musical distance" (key invariant)[2][3]. For example, the transition $C \to D\#m$ will share the same label as $F \to G\#m$ and its enharmonic $C \to E\flat m$, i.e.,

$$(C, D\#m) = (C, E\flat m) = (F, G\#m) = \lambda_k \tag{5.29}$$

By sequentially reading the obtained interval sequence $x = \{\lambda_1, \lambda_2, ..., \lambda_1, ..., \lambda_l\}$, we will extract a directed graph G (see Table 5.5) where each node represents a chord transition or interval ($n = |\{\lambda_i\}|$), and each interval transition is represented by an edge ($m = |\{\lambda_i \to \lambda_j\}|$).

5.6.2 Kernel function

Based on the algorithm from Horváth et al. (2004), we build a kernel which takes any two interval graphs from the input space, extracts their simple cycles to build a feature space, and computes a similarity as the weighted inner product in the feature space. In our case, the mapping function Φ is defined

[2]For the sake of consistency we have not made the distinction between ascending or descending intervals.

[3]Please note that the chord type (minor, major, diminished, etc.) is already incorporated in the graph representation through the λ values, e.g., $(Cdim, Am) = (Edim, C\#m) = \lambda_k$.

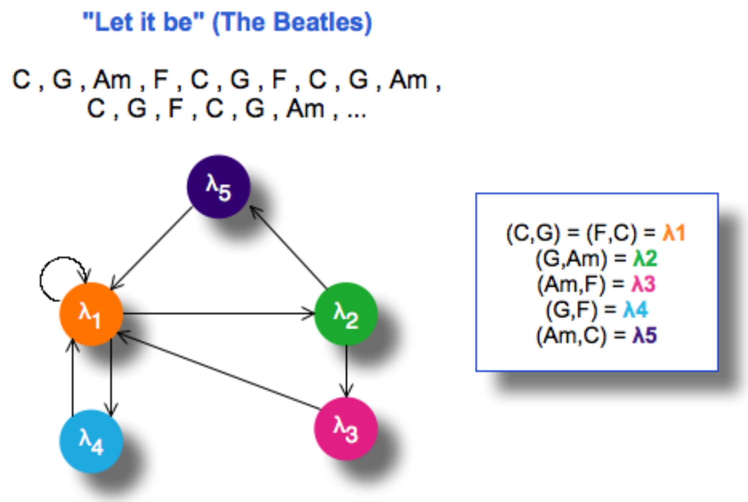
Chords	C,G,Am,F,C,G,F,C,G,Am,C,G,F,C,G,Am,...
Labels	$(C, G) = (F, C) = \lambda_1$, $(G, Am) = \lambda_2$ $(Am, F) = \lambda_3$, $(G, F) = \lambda_4$, $(Am, C) = \lambda_5$
Intervals	$\lambda_1, \lambda_2, \lambda_3, \lambda_1, \lambda_1, \lambda_4, \lambda_1, \lambda_2, \lambda_5, \lambda_1, \lambda_4, \lambda_1, \ldots$
Graph	 "Let it be" (The Beatles) C , G , Am , F , C , G , F , C , G , Am , C , G , F , C , G , Am , ... λ5 λ1 λ2 λ3 λ4 (C,G) = (F,C) = λ1 (G,Am) = λ2 (Am,F) = λ3 (G,F) = λ4 (Am,C) = λ5

Table 5.5: Transformation of an extract of the chord progression of "Let it be" from The Beatles into an interval graph.

as a mapping to the set of all possible simple cycles of the graph

$$G \to \Phi_{\text{SC}}(G) = \mathcal{S}(G) \tag{5.30}$$

where the set S is represented in a vector space whose dimensions are indicating the presence or absence of each element in the set. For a particular graph G_j, its feature vector has entries $[\phi(G_j)]_i$ which are equal to 1 if the simple cycle with index i (denoted as cycle i in the sequel) is present in the graph and 0 otherwise. We then compute the kernel function as the weighted inner product between the feature vectors (simple cycles vector) corresponding to the two objects x and y:

$$k(x, y) = \langle \phi(x), \phi(y) \rangle_{\tilde{W}} = \phi(x)^T \mathbf{D} \phi(y) \tag{5.31}$$

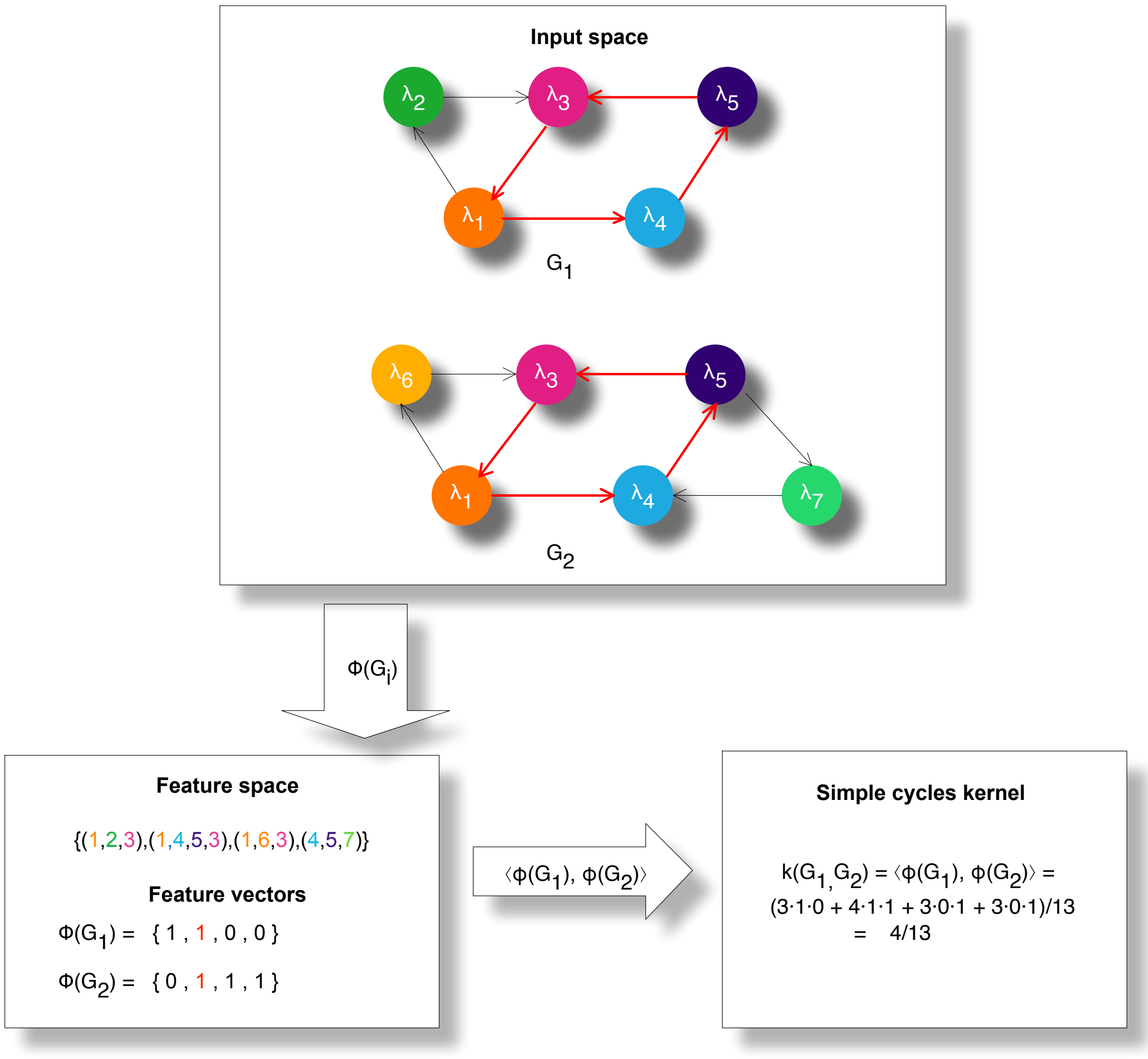


Figure 5.7: Computation of the simple-cycles weighted kernel on two initial graphs, G_1 and G_2.

where $\mathbf{D}$ is the normalized diagonal weight matrix

$$[D]_{ii} = d_{ii} = \frac{w_i}{\sum_{j \in \mathcal{S}(G_k) \cup \mathcal{S}(G_l)} w_j} \tag{5.32}$$

and w_i is the length of the i-th cycle. The motivation for this weighting is to favor longer cycles, so that two graphs sharing a long cycle are considered as more similar as two graphs sharing one short cycle. Furthermore, the kernel weights are normalized by dividing them by their sum. The complete procedure is described in Algorithm 9 and an example on how to compute the weighted kernel is given in Figure 5.7.

The complexity of this kernel is the same as for Horváth et al. (2004) when using the approach from Read and Tarjan (1975), i.e., $O(v + e(c + 1))$ where v is the number of vertices, e is the number of edges, and c is the number of simple cycles.

The code of the simple-cycles weighted kernel is available at `http://github.com/silviagdiez/thesis`.

Algorithm 9 Simple-cycles Weighted Kernel: computation of the kernel matrix.

```
Input:
  • $maxL > 0$: maximum length of extracted simple cycles.
  • $s_1, ..., s_r$: list of chord sequences to be compared.
Output:
  • $K$: the Simple-cycles Weighted Kernel matrix.
1. for $k$,$l = 1$ to $r$ do
2.     Transform chord sequences $s_k$ and $s_l$ into directed labeled graphs $G_k$
       and $G_l$ following the procedure from Table 5.5.
3.     Extract all simple cycles of length $< maxL$, $\mathcal{S}(G_k)$ and $\mathcal{S}(G_l)$, from
       $G_k$ and $G_l$, with the algorithm described in Read and Tarjan (1975) or
       Ponstein (1966).
4.     Create the feature vectors, $\phi(G_k)$ and $\phi(G_l)$, of length $|\mathcal{S}(G_k) \cup \mathcal{S}(G_l)|$,
       whose entry $[\phi(G_k)]_i = 1$ if the $i$-th cycle is in $\mathcal{S}(G_k)$ and 0 otherwise.
       Idem for $\phi(G_l)$.
5.     For all the cycles of $\mathcal{S}(G_k) \cup \mathcal{S}(G_l)$, compute the corresponding elements
       $i$ of the diagonal matrix $D$ from Equation (5.32).
6.     Compute $[K]_{kl} = \phi(G_k)^T D \phi(G_l)$.
7. end for
```

5.7 Web mining for chord data

A complete Information Retrieval (IR) system (see Section 5.2 for more details) has been developed and is presented in this section. In order to evaluate the proposed method, we have built a web retrieval system which contains a web crawler for mining chord data, as well as a model based on the Space Vector model, as it applies the cyclic patterns kernel. This IR system belongs to the query by example category, as users will provide a song, in order to find the most similar ones.

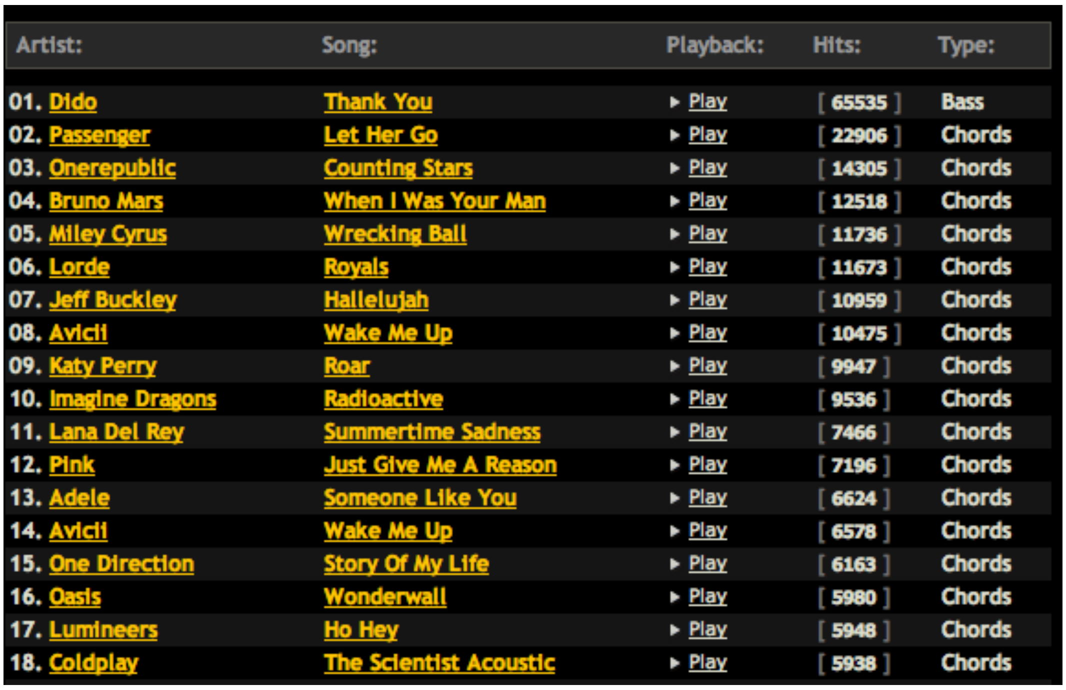
For the empirical validation of the retrieval performance of our kernel, we have gathered chord data from various sources, one of which is a web site for guitar chord tabs www.ultimate-guitar.com. In order to retrieve this data from the web, a Java web crawler designed to gather all chord data from www.ultimate-guitar.com has been implemented. This crawler takes into account the politeness rules of normal web crawlers, in order to avoid overloading the server.

The web crawler takes as seed the home page of www.ultimate-guitar.com and an input parameter which is the number of pages of songs that we want to crawl. It then starts visiting all links with song titles as shown in Figure 5.8.

Every link redirects us to another page where all the available versions of that song are listed, e.g.,Figure 5.9.

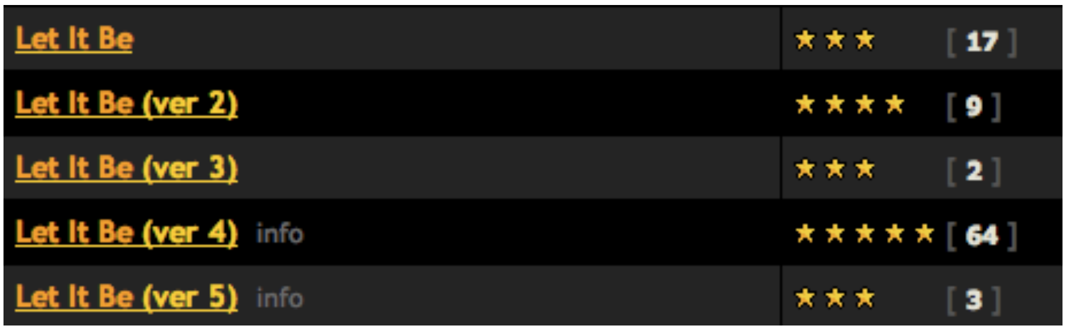
For our testing purposes, the crawler will only visit the links of those versions that are 5 star-rated with at least 5 votes, e.g., in the example of Figure 5.9 only version 4 would be extracted. Each link of the selected version is then crawled, and all chord content is parsed from the HTML file. Eventually, a file with the chord sequence for all songs is created.

The code of the crawler is available at `http://github.com/silviagdiez/thesis`.



Artist:	Song:	Playback:	Hits:	Type:
01. Dido	Thank You	▸ Play	[65535]	Bass
02. Passenger	Let Her Go	▸ Play	[22906]	Chords
03. Onerepublic	Counting Stars	▸ Play	[14305]	Chords
04. Bruno Mars	When I Was Your Man	▸ Play	[12518]	Chords
05. Miley Cyrus	Wrecking Ball	▸ Play	[11736]	Chords
06. Lorde	Royals	▸ Play	[11673]	Chords
07. Jeff Buckley	Hallelujah	▸ Play	[10959]	Chords
08. Avicii	Wake Me Up	▸ Play	[10475]	Chords
09. Katy Perry	Roar	▸ Play	[9947]	Chords
10. Imagine Dragons	Radioactive	▸ Play	[9536]	Chords
11. Lana Del Rey	Summertime Sadness	▸ Play	[7466]	Chords
12. Pink	Just Give Me A Reason	▸ Play	[7196]	Chords
13. Adele	Someone Like You	▸ Play	[6624]	Chords
14. Avicii	Wake Me Up	▸ Play	[6578]	Chords
15. One Direction	Story Of My Life	▸ Play	[6163]	Chords
16. Oasis	Wonderwall	▸ Play	[5980]	Chords
17. Lumineers	Ho Hey	▸ Play	[5948]	Chords
18. Coldplay	The Scientist Acoustic	▸ Play	[5938]	Chords

Figure 5.8: Page with list of songs in www.ultimate-guitar.com.

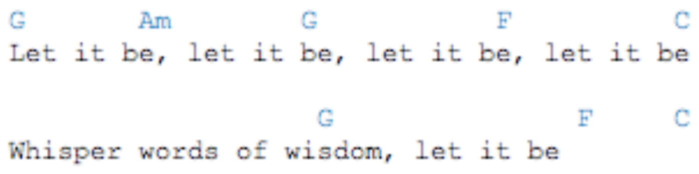


Let It Be	★★★	[17]
Let It Be (ver 2)	★★★★	[9]
Let It Be (ver 3)	★★★	[2]
Let It Be (ver 4) info	★★★★★	[64]
Let It Be (ver 5) info	★★★	[3]

Figure 5.9: Page with all available versions of a song in www.ultimate-guitar.com.

5.8 Empirical testing

For our testing purpose, two different tasks are evaluated: (i) a cover song retrieval task, and (ii) an idiom retrieval task. We first present the data used in the experiments, as well as the chosen lexicon. Our simple-cycles weighted kernel method is compared to several measures from string matching, as well as graph comparison techniques.



```
G       Am        G            F          C
Let it be, let it be, let it be, let it be

                   G               F     C
Whisper words of wisdom, let it be
```

Figure 5.10: Page with all available versions of a song in www.ultimate-guitar.com.

The *cover song data set* has been extracted from two different sources: the Beatles chord annotations from the Isophonics[4] database (Queen Mary, University of London), and the user-generated chord files from the Ultimate-Guitar[5] database.

Although our first source of chord progressions has already been used in Music Information Retrieval (MIR), we are the first to exploit, to the best of our knowledge, a popular Internet guitar's chord database for similarity retrieval. The Ultimate-Guitar database contains more than 250,000 user-generated sequences of guitar's chords of popular pop/rock music. Although several versions are available for each of the Beatles' songs, only well-ranked songs have been extracted (5 star rated songs with at least 5 votes), making a total of 71 songs.

These same songs have been extracted afterwards from the Isophonics database, forming 71 classes of two songs each (142 songs in total), where the songs from the Isophonics database are used as query over the remaining 71 songs from the Ultimate-Guitar database (one relevant song per query). Although there exists a well-known MIREX audio cover song task, this evaluation task takes audio signals as input while our work is centered on chords, so that it cannot be applied here.

The *idiom data set* has been fully extracted from the Ultimate-Guitar database and contains 296 songs partitioned in two classes (101 songs for the first class, sharing a common 4-chords idiom[6], and 195 songs for the second

[4]`isophonics.net`
[5]`www.ultimate-guitar.com`
[6]The sequence "C,G,Am,F" is considered as an idiom in modern pop/rock composition.

class).

Both data sets are available at `http://github.com/silviagdiez/thesis`.

In both cases, a modest lexicon containing all major and minor root chords (flat and sharp) has been used. We believe that this choice is representative enough for our purpose, while avoiding bad transcription issues from users in the Ultimate-Guitar database, e.g., the chord $C5$ appears instead of C.

We have chosen to use the Isophonics database as it was used in previous studies for harmonic similarity and was easily available. Although the Ultimate-Guitar data had to be crawled, we wanted to exploit this vast source of harmonic progressions and tackle the difficulties of working with real, noisy, data. This data was only used in the context of our work, and all copyright rights remain with Ultimate-Guitar.

Please note that the kernel from Horváth et al. (2004) has not been used for comparison, as this kernel focuses on simple cycles and trees of a graph. As we want to exploit simple cycles of a graph, we consider that including tree motifs would only add noise to the final measure. For this reason, it does not appear as one of the compared methods.

5.8.1 Cover song retrieval task

Cover song retrieval (see for instance Bello (2007)) is a popular task in MIR which aims at identifying the versions of a given song. For this purpose, the cover song data set described above has been used. We query the Ultimate-Guitar database with each song from the Isophonics chord annotation (the "query song"), providing a ranking of the Ultimate-Guitar songs in decreasing order of similarity with the Isophonics query song (please see Section 5.8 for more details).

This is particularly interesting for the case of Ultimate-Guitar, where we have seen that some users deliberately omit repeating portions of a song for practical reasons. We want to test our method with other more traditional

It appears in songs such as *Let it be* (The Beatles), and *With or without you* (U2).

approaches which take the full length sequence into account, and where this type of behavior can significantly affect the retrieval.

The *average ranking* position of all retrieved songs, as well as two popular recall measures (de Haas et al., 2008) describing the accuracy of our method have been chosen to describe the performance of our method in a retrieval task, i.e., ranking of cover songs should be higher than non-cover songs.

These measures are reported in Table 5.6: the *average first tier* (the number of correctly retrieved songs among the best $(n_c - 1)$ matches divided by $(n_c - 1)$ with n_c the class size, i.e., in our case $n_c = 2$), and the *average second tier* (number of correctly retrieved songs among the best $(2n_c - 1)$ matches divided by $(n_c - 1)$).

In order to compare our method to other baseline methods, the same methodology[7] has been applied to three string matching techniques – the edit distance and longest common subsequence widely used in sequence matching (see, e.g., Gusfield (1997)) and the all-subsequences kernel (Shawe-Taylor and Cristianini, 2004) which is an efficient method that compares all sub-sequences of two strings –, and a graph comparison kernel – the fast sub-tree kernel, a similarity measure between graphs that is fast to compute and that outperforms other graph kernels (Shervashidze and Borgwardt, 2009). For methods needing a parameter, the fast sub-tree kernel and the simple-cycles kernel, we have chosen a maximum cycle length (tree depth) of 7 – longer cycles or deeper trees become too song-specific, and are not of interest for us.

Please note that the edit distance and the longest common subsequence have been normalized and centered as explained in Annex B.

Although chosen baseline methods may appear simplistic, our aim is to compare our algorithm with a variety of methods under the same conditions. Purpose-built methods using different chord representations, or needing parameter tuning are not compared in the present work for obvious reasons of adaptation, leaving this task for further work.

[7]Interval sequences have been provided as input for each baseline method, so that all compared methods are transposition invariant and evaluated under similar conditions.

Although results show no improvement for the first tier, and just a slight improvement of the second tier (see average first and second tier in Table 5.6) from the baseline methods, there is a clear improvement in the average general ranking of retrieved songs. These results are encouraging for using the Ultimate-Guitar database as a future source for chord progression data.

5.8.2 Idiom retrieval task

Idioms have recently attracted the attention of MIR as a new object of musicological interest. An *idiom* is defined by Mauch et al. (2007) as a "prominent chord sequence in a particular style, genre or historical period". Users have also discovered this notion of idiom as shown in a youtube video[8], where a sequence of 4 chords is used to assemble the melody of several pop/rock songs. Interestingly, people who liked a few of these songs tended to also appreciate the others.

We have tried to recover the songs containing the idiom "C,G,Am,F" (or "I-V-VI-IV") in a supervised learning task, i.e., does the song contain the idiom or not? Please note that the purpose is not merely to retrieve this particular idiom, which is a trivial task, but in a wider sense we wish to group songs that contain similar idioms, without knowing in advance which ones.

We have measured the error rates of all methods and compared them by applying a 10-fold double cross validation with an RBF SVM on the idiom data set from the Ultimate-guitar web site. Classification rates with a 95% confidence interval are reported in Table 5.7. These results show an increase of performance of our method of 7% from the closest base-line method.

Please note that the edit distance and the longest common subsequence have been normalized and centered as explained in Annex B.

[8]`http://www.youtube.com/watch?v=qHBVnMf2t7w`

Similarity	**First tier average**	**Second tier average**	**Average ranking**
Edit distance	78.87% ± 6.76	87.32% ± 5.51	4.169 ± 1.64
Longest common subs.	60.56% ± 8.10	69.01% ± 7.66	8.662 ± 2.59
All-subsequence kernel	28.17% ± 7.45	43.66% ± 8.22	15.929 ± 3.32
Fast sub-tree kernel	52.11% ± 8.27	61.97% ± 8.04	11.943 ± 2.78
Simple-cycles kernel	78.87% ± 6.76	88.73% ± 5.24	2.915 ± 1.09

Table 5.6: Average first tier, second tier, and average ranking for the cover retrieval task with 95% confidence intervals.

Similarity	**Classification rate and confidence interval**
Edit distance	68.56% ± 1.53
Longest common subsequence	69.91% ± 2.41
All-subsequence kernel	68.56% ± 1.53
Fast sub-tree kernel	81.06% ± 4.22
Simple-cycles kernel	88.50% ± 2.02

Table 5.7: Classification rates with an SVM with a 95% confidence interval for the idiom retrieval task.

5.9 Conclusion

In this chapter we have introduced a simple-cycle similarity method based on the harmonic progression of a song. We have presented the notions of a theoretically well-founded method, and shown its applicability to our problem. This approach has furthermore been validated on an idiom and a cover song retrieval task. The obtained results suggest the usefulness of extracting repetitive sub-structures for music similarity purposes by means of a simple-cycles weighted kernel. Further work will try to improve the presented algorithm by performing an approximate cycle matching, and by replacing labels by musical distances between chords.

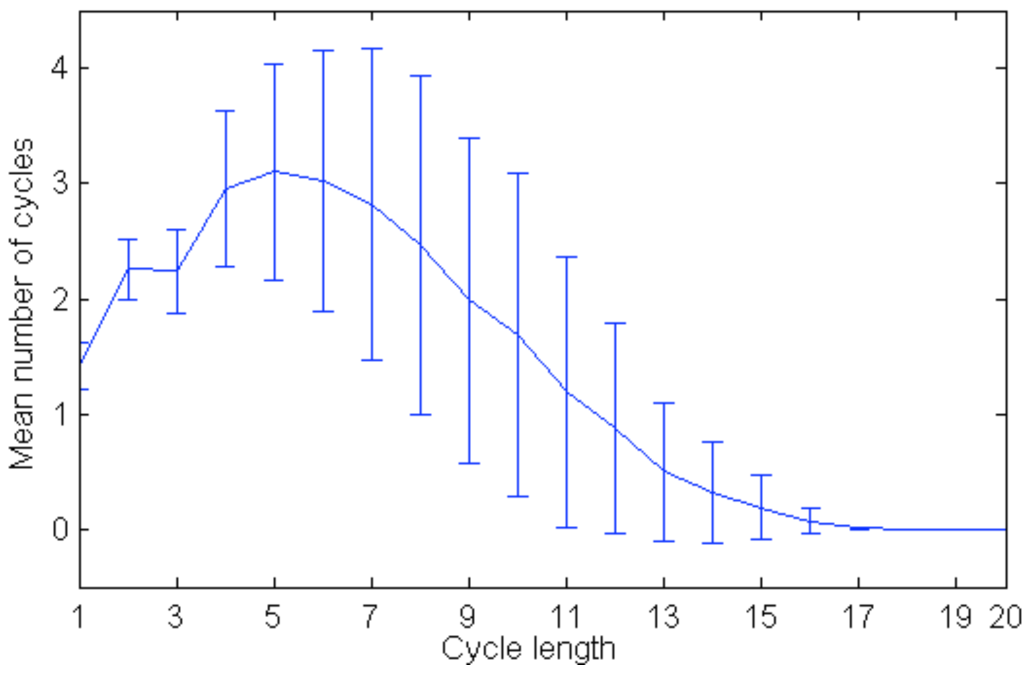


Figure 5.11: Error bar showing the average number of simple cycles per song and per cycle length of our chord progressions data. 95% confidence intervals are also shown.

Chapter 6

General conclusion

This last chapter provides a summary of the presented work in the different chapters. As already explained in Chapter 1, randomization can be of use in multiple applications such as:

- computing robust similarity measures between two sequences by allowing some path randomization on the editing paths,
- improving the artificial intelligence on two-player zero-sum games, by allowing a more human-like way of randomizing strategies defined in the game tree,
- defining smooth random trajectories for motion planning on a continuous environment.

Some of the presented techniques outperform the existing ones, and some other present a different approach for solving existing problems based on various extensions of the RSP framework on sequence comparison, game theory, and motion planning.

The last chapter, which is not based on the RSP framework, applies the principles from information retrieval and graph kernels for information extraction based on cycles for music comparison. It computes a similarity measure for harmonic sequences of songs thanks to the extraction of cycles in a graph representation of songs.

6.1 Contributions and limitations

In this section we present an overview of the contributions and limitations of each of the techniques presented in this thesis.

Chapter 1 introduces an alternative, more intuitive, derivation of the forward and backward variables in the context of the Sum-over-Paths framework, and its relation to the main quantities of interest.

Chapter 2 introduces four novel methods based on the Sum-over-Paths framework for Approximate String Matching (ASM): (i) the Sum-over-Paths edit distance, (ii) the Sum-over-Paths common subsequences, (iii) the Normalized Sum-over-Paths edit distance, and (iv) the Normalized Sum-over-Paths common subsequences.

These measures provide a model-independent technique for computing similarity by taking all alignments into account. They also avoid noisy measures by favoring relevant subsequences of nearly-optimal alignments. Furthermore, their normalized versions overcome normalization issues. These measures have been proven to outperform other ASM techniques through empirical validation.

However, they are not valid kernels, as the derived similarity matrix is not positive semi-definite. Another limitation is that the Sum-over-Paths Edit Distance is not a distance but rather a dissimilarity measure (see Appendix A for more details). Further work will focus on defining a valid kernel over the same principles.

A final point that has to be tested over large graphs, is how the SoP distances are affected by an increasing number of edges connecting the nodes. This issue was introduced in Section 2.3.1, and makes the probabilities of reaching a destination node dependent on the number of edges when the graph becomes large enough. The SoP distances mitigate this issue by favoring suboptimal paths, but the relation between the spread entropy and the size of the graph has to be studied.

Chapter 3 provides an optimal randomized policy based on the Sum-over-Paths framework for solving continuous-state path planning problems with multiple sources and multiple goals. However, the main drawback of this model is that it assumes that paths are uncorrelated, which is hardly the case in practice.

On the other hand, it introduces a diffusion parameter for controlling the trade-off between exploration and exploitation, and it shows some interesting links between biased random walks on a graph (discrete RSP) and continuous-state Feynman-Kac diffusion processes. Further work will study the possibility of introducing a mass parameter (inertia) for smoothing the trajectories, and therefore avoiding abrupt changes in direction.

Chapter 4 provides a novel global optimal strategy based on the Sum-over-Paths framework given a level of entropy for simulating the AI in two-player zero-sum games. Although the notion of entropy has been widely used for controlling randomness in AI, this new method spreads the entropy over full strategies or paths, instead of single moves. In this case as well, the main drawback of this method is that paths are assumed to be uncorrelated. Furthermore, the opponent is assumed to be fully rational, which is not realistic for some problems.

Future work will investigate the extension of the $\mathcal{R}$minimax to multi-player games as well as online or dynamic games. Indeed, we could adapt our equations to take into account more than one player for other n-player games. We could achieve this by adding a dummy (n+1)-player who can only take one action, and whose reward is such that the sum remains zero for the game at any state.

Furthermore, we could create an adaptive version of the $\mathcal{R}$minimax, which estimates the θ parameter of a real player, and adapts the strength of its game accordingly. This would allow to mimic the users' behavior by following a similar learning curve, which would improve the playing experience.

Eventually, we would like to apply this framework to nested Monte-Carlo search techniques, which are well-known heuristic search algorithms very popular in game playing.

Chapter 5 provides a similarity measure for songs based on the repetitive harmonic features of songs. This similarity measure deals with large structural changes in chord progression which are extracted as cycles from a graph and is computed by means of a kernel function. This approach for harmonic comparison is new in the domain. It also exploits a novel source of user-generated data that is readily available on the Internet.

Further work will try to improve the presented algorithm by performing an approximate cycle matching, which would allow retrieving songs which share similar, but not identical, cycles. Furthermore, we could try to differentiate between cycles and its connectors, e.g., cyclic patterns may be linked by linear structures, which are equally important for harmony resolution. Indeed, we have noticed that by creating the graph structure of the harmonic sequence, some information is lost (such as those cycle connectors), while some new cycles have been added. Both issues could be solved by identifying cycles and connectors separately.

We would also like to try the replacement of labels denoting the intervals by a more accurate musical notion, such as the Tonal Pitch distance from de Haas et al. (2008). Having the real musical distance could allow for fuzzy search, where instead of matching exact intervals, we can match intervals with similar distances.

In the presented work, we have used a vector space model for Information retrieval, but we could apply some of the probabilistic approaches, which have already proven to be useful in text retrieval. A first model is the Inference Network model (Turtle and Croft, 1992) which introduces the notion of "concept" as semantic information built on top of the terms. In our case, we could think that harmony contains as well a higher level of abstraction built on top of simple cycles and connectors, and this would allow for a more accurate ranking of songs. This method matches the songs and the query by means of those underlying concepts. By fixing the prior knowledge on certain songs, we could boost songs with higher scores, so that we improve the general quality of the matching.

But we could also test the Language Model (see, e.g., Moens (2006)) and adapt it to music similarities. This paradigm assumes that each document, or song in our case, has an underlying model from which the query can be generated. By computing the probabilities that the query is generated by each of the songs or models, we obtain a ranking of the different songs. Again, we can choose to use the simple cycles as the terms on which to build the probability distributions, or we can try to identify a better descriptor of a song.

6.2 Global discussion

This thesis has aimed at proving the applicability of the Randomized Shortest Path in several domains. Indeed, by allowing a certain degree of randomization on the paths defined on a graph, we have seen how to build robust measures for string comparison, smooth randomized trajectories for path planning, and random strategies in Artificial Intelligence that mimic human behavior.

Although the RSP framework may be applied to many other fields, one preliminary condition is needed: the data must be representable in form of graph, and the paths defined on that graph must represent some information that can benefit from randomization.

This is not the case of Chapter 5, where data was presented in form of harmonic sequences and it was transformed into a graph structure. Unlike previous chapters, we present here a technique which is not based on the RSP framework, but rather on motif extraction from graphs. Indeed, randomization was of little use in this case.

We are certain that the RSP can be the basis for many other techniques, including music comparison, as the experimental results have already proven. We remain confident to see an increasing number of approaches which take advantage of randomized paths, or other graph structures.

Appendix A

Proof that the SoPED is not a distance

In order to proof that the SoP edit distance does not comply with the triangle inequality, and is thus not a distance but a dissimilarity measure, let us take the graph from Figure A.1 as example. We are going to prove that the triangle inequality does not hold, first, for the cases where $\theta \to 0$, where probabilities are uniformly distributed, and later for $\theta \to \infty$, where the probabilities are peaked on the shortest path. Local costs $c_{k,k'}$ are assumed to be unitary for all edges.

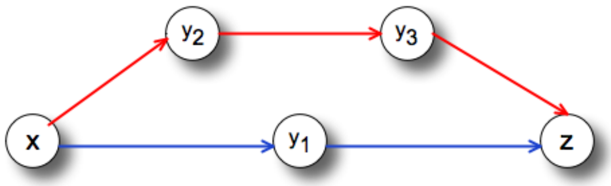


Figure A.1: Example graph for proof of triangle inequality. Source node, x, is tied to destination node, z, by two paths: an optimal shortest path $\wp_1$ of length two (blue) and a sub-optimal path $\wp_2$ of length three (red).

Proof. Case where $\theta \to 0$:

- Path probabilities are $\mathrm{P}(\wp_1) = \mathrm{P}(\wp_2) = \frac{1}{2}$.

$$\begin{aligned}
d(x,z) =& \mathbb{E}[C] = \sum_{\wp_1,\wp_2} \mathrm{P}(\wp)\, C(\wp) = \frac{1}{2}[c_{x,y_1} + c_{y_1,z}] + \frac{1}{2}[c_{x,y_2} + c_{y_2,y_3} + c_{y_3,z}] \\
=& \frac{1}{2} \cdot 2 + \frac{1}{2} \cdot 3 = 5/2 \\
d(x,y_2) + d(y_2,z) =& d(x,y_3) + d(y_3,z) = d(x,y_2) + d(y_2,y_3) + d(y_3,z) \\
=& \mathrm{P}(\wp_{x,y_2}) \cdot c_{x,y_2} + \mathrm{P}(\wp_{y_2,y_3}) \cdot c_{y_2,y_3} + \mathrm{P}(\wp_{y_3,z}) \cdot c_{y_3,z} \\
=& 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 = 6/2 \\
d(x,y_1) + d(y_1,z) =& \mathrm{P}(\wp_{x,y_1}) \cdot c_{x,y_1} + \mathrm{P}(\wp_{y_1,z}) \cdot c_{y_1,z} \\
=& 1 \cdot 1 + 1 \cdot 1 = 4/2 \\
\Rightarrow& d(x,z) > d(x,y_1) + d(y_1,z)
\end{aligned}$$

Case where $\theta \to \infty$:

- Path probabilities are $\mathrm{P}(\wp_1) = 1$ and $\mathrm{P}(\wp_2) = 0$.

$$\begin{aligned}
d(x,z) =& \mathbb{E}[C] = \sum_{\wp_1,\wp_2} \mathrm{P}(\wp)\, C(\wp) = 1 \cdot [c_{x,y_1} + c_{y_1,z}] + 0 \cdot [c_{x,y_2} + c_{y_2,y_3} + c_{y_3,z}] \\
=& 1 \cdot 2 = 2 \\
d(x,y_2) + d(y_2,z) =& d(x,y_3) + d(y_3,z) = d(x,y_2) + d(y_2,y_3) + d(y_3,z) \\
=& \mathrm{P}(\wp_{x,y_2}) \cdot c_{x,y_2} + \mathrm{P}(\wp_{y_2,y_3}) \cdot c_{y_2,y_3} + \mathrm{P}(\wp_{y_3,z}) \\
=& 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 = 3 \\
d(x,y_1) + d(y_1,z) =& \mathrm{P}(\wp_{x,y_1}) \cdot c_{x,y_1} + \mathrm{P}(\wp_{y_1,z}) \cdot c_{y_1,z} \\
=& 1 \cdot 1 + 1 \cdot 1 = 2 \\
\Rightarrow& d(x,z) \leq d(x,y) + d(y,z)\ \forall x,y,z
\end{aligned}$$

□

It is clear from the proof that the triangle inequality does not hold for small values of θ. However, when $\theta \to \infty$, the SoPED becomes a distance.

The reason behind this behavior, is that we recover the typical edit distance by taking only the shortest, optimal, paths.

Appendix B

Normalization of kernel matrices

B.1 Kernel centering

We applied the transformation $\mathbf{K_c} = \mathbf{HKH}$ where $\mathbf{K}$ is the kernel or the similarity matrix, $\mathbf{H} = (\mathbf{I} - \mathbf{ee^T}/n)$ is the centering matrix, $\mathbf{I}$ is the identity matrix, and $\mathbf{e}$ is a column vector full of ones.

Notice that for $\mathbf{K}_{\mathrm{ED}}^{\mathrm{SoP}}$, $\mathbf{K}_{\mathrm{LED}}$ and $\mathbf{K}_{\mathrm{SED}}$, which are dissimilarity measures, the conversion to a similarity measure – as well as centering – is achieved through $\mathbf{K_c} = -\frac{1}{2}\mathbf{H\Delta H}$ where $\mathbf{\Delta}$ is the distance matrix containing the edit distances. This is a classical multidimensional scaling procedure Borg and Groenen (1997); Cox and Cox (2001).

B.2 Kernel normalization

After centering, the obtained similarity matrices may be further normalized with $\mathbf{K_n} = \mathbf{D}^{-1/2}\mathbf{K_c}\mathbf{D}^{-1/2}$ where $\mathbf{D}$ is a diagonal matrix containing the elements of the diagonal of $\mathbf{K_c}$, $\mathbf{D} = \mathbf{Diag}(\mathbf{K_c})$.

Bibliography

Y. Achbany, F. Fouss, L. Yen, A. Pirotte, and M. Saerens. Optimal tuning of continual exploration in reinforcement learning. *Proceedings of the 16th International Conference on Artificial Neural Networks (ICANN 06). Lecture notes in Computer Science*, 4131:734–749, 2006.

Y. Achbany, F. Fouss, L. Yen, A. Pirotte, and M. Saerens. Tuning continual exploration in reinforcement learning: An optimality property of the Boltzmann strategy. *Neurocomputing*, 71:2507–2520, 2008.

G. M. Adelson-Velsky, V. L. Arlazarov, and M. V. Donskoy. *Algorithms for games*. Springer-Verlag New York, Inc., 1988.

T. Akamatsu. Cyclic flows, Markov process and stochastic traffic assignment. *Transportation Research B*, 30(5):369–386, 1996.

J. C. Amengual and E. Vidal. On the estimation of error-correcting parameters. In *Proceedings of the International Conference on Pattern Recognition (ICPR)*, pages 2883–2886, 2000.

A. Arenas, A. Fernandez, S. Fortunato, and S. Gomez. Motif-based communities in complex networks. *Journal of Physics A: Mathematical and Theoretical*, 41(22):224001, 2008.

G. Arfken and H. Weber. *Mathematical methods for physics*. Elsevier, 2005.

V. L. Arlazarov, E. A. Dinic, M. A. Kronod, and I. A. Faradzev. On economical construction of the transitive closure of an oriented graph. *Doklady Akademii Nauk SSSR*, 194(11):1209–1210, 1970.

R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval - the concepts and technology behind search, Second edition.* Pearson Education, 2011.

L. R. Bahl and F. Jelinek. Decoding for channels with insertions, deletions, and substitutions with applications to speech recognition. *IEEE Transactions on Information Theory*, 21(4):404–411, 1975.

J. P. Bello. Audio-based cover song retrieval using approximate chord sequences: Testing shifts, gaps, swaps and beats. In *Proceedings of the 8th International Society for Music Information Retrieval Conference (ISMIR)*, pages 239–244, 2007.

H. Berg. *Random walks in biology.* Princeton University Press, 1993.

D. P. Bertsekas. *Dynamic Programming and Optimal Control.* Athena Scientific, 2000.

F. Beuvens and T.Dullier. Développement et expérimentation d'une plateforme de reconnaissance de gestes par stylet. Master's thesis, Université Catholique de Louvain, 2009.

I. Borg and P. Groenen. *Modern multidimensional scaling: Theory and applications.* Springer, 1997.

S. Boyd, P. Diaconis, and L. Xiao. Fastest mixing markov chain on a graph. *SIAM Review*, 46(4):667–689, 2004.

M. Brand. A random walks perspective on maximizing satisfaction and profit. In *Proceedings of the 2005 SIAM International Conference on Data Mining*, pages 12–19, 2005.

P. Bucher and K. Hofmann. A sequence similarity search algorithm based on a probabilistic interpretation of an alignment scoring system. In *Proceedings of the Fourth International Conference on Intelligent Systems for Molecular Biology (ISMB)*, volume 4, pages 44–51, 1996.

M. Buro. Toward opening book learning. In *Proceedings of the Workshop on Computer Games*, pages 1–5, 1997.

N. Cancedda, E. Gaussier, C. Goutte, and J. M. Renders. Word sequence kernels. *Journal of Machine Learning Research*, 3:1059–1082, 2003.

D. Carmel and S. Markovitch. Exploration strategies for model-based learning in multi-agent systems: Exploration strategies. *Autonomous Agents and Multi-Agent Systems*, 2(2):141–172, 1999.

T. Cazenave. Nested monte-carlo search. In *Proceedings of the International Joint Conferences on Artificial Intelligence*, pages 456–461, 2009.

M. Chaichian and A Demichev. *Path integrals in physics, vol 1: stochastic processes and quantum mechanics.* Institute of Physics Publishing, 2001.

G. Chaslot. Monte-carlo tree search. Master's thesis, 2010.

L. Chatriot, S. Gelly, J.B. Hoock, J. Pérez, A. Rimmel, and O. Teytaud. Introduction de connaissances expertes en bandit-based monte-carlo planning avec application au computer-go. In *Proceedings of the Journées Francophones de Planification, Décision et Apprentissage pour la conduite de systèmes*, 2008.

C.I. Connolly, J.B. Burns, and R. Weiss. Path planning using laplace's equation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 3, pages 2102–2106, 1990.

J. Cook. Basic properties of the soft maximum. Unpublished manuscript available from www.johndcook.com/blog/2010/01/13/soft-maximum, 2011.

P.Arnaud. Coquelin and R. Munos. Bandit algorithms for tree search. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, pages 67–74, 2007.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms, 2nd edition.* MIT Press, McGraw-Hill Book Company, 2000.

T. M. Cover and J. A. Thomas. *Elements of information theory.* John Wiley & Sons, 1991.

T. Cox and M. Cox. *Multidimensional scaling, 2nd edition.* Chapman and Hall, 2001.

K. Dautenhahn and H. Cruse. Computer simulations of path generation and path form modification with local rules working on a parallel cell-based architecture. *Computers & Mathematics with Applications*, 28(5):75 – 88, 1994.

W. Bas de Haas, R. C. Veltkamp, and F. Wiering. Tonal pitch step distance: a similarity measure for chord progressions. In *Proceedings of the 9th International Society for Music Information Retrieval Conference (ISMIR)*, pages 51–56, 2008.

W. Bas de Haas, M. Rohrmeier, R. C. Veltkamp, and F. Wiering. Modeling harmonic similarity using a generative grammar of tonal harmony. In *Proceedings of the 10th International Society for Music Information Retrieval Conference (ISMIR)*, pages 549–554, 2009.

P. Del Moral. *Feynman-Kac formulae: genealogical and interacting particle systems with applications.* Springer, 2004.

I. Deliège, M. Mélen, D. Stammers, and I. Cross. Musical schemata in real time listening to a piece of music. *Music Perception*, 14(2):117–160, 1996.

J. C. Delvenne and A. S. Libert. Centrality measures and thermodynamic formalism for complex networks. *Physical Review E*, 83(4):046117, 2011.

C. Ding and X. He. Linearized cluster assignment via spectral ordering. In *Proceedings of the twenty-first international conference on Machine Learning (ICML)*, pages 225–232, 2004.

C.B. Do, S.S. Gross, and S. Batzoglou. CONTRAlign: Discriminative training for protein sequence alignment. In *Research in Computational Molecular Biology*, pages 160–174, 2006.

P. G. Doyle and J. L. Snell. *Random Walks and Electric Networks.* The Mathematical Association of America, 1984.

C. Drake. Psychological processes involved in the temporal organization of complex auditory sequences: Universal and acquired processes. *Music Perception*, 16(1):11–26, 1998.

R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis: Probabilistic models of proteins and nucleic acids.* Cambridge University Press, 1998.

L. Ekroot and T. Cover. The entropy of markov trajectories. *IEEE Transactions on Information Theory*, 39(4):1418–1421, 1993.

R. E. Fan, K. W. Chang, C. J. Hsieh, X. R. Wang, and C. J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

G. D. Fornay. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

K. Francoisse, I. Kivimaki, A. Mantrach, F. Rossi, and M. Saerens. A bag-of-paths framework for network data analysis. *Submitted for publication; available on ArXiv as ArXiv:1302.6766*, pages 1–36, 2013.

S. García-Díez, F. Fouss, M. Shimbo, and M. Saerens. A sum-over-paths extension of edit distances accounting for all sequence alignments. *Pattern Recognition*, 44(6):1172–1182, 2011.

C. W. Gardiner. *Handbook of stochastic methods.* Springer, 2002.

T. Gärtner. *Kernels For Structured Data.* World Scientific Publishing, 2009.

G. Gigerenzer and R. Selten. *Bounded rationality: the adaptive toolbox.* MIT Press, 2002.

V. Girardin. Entropy minimization for Markov and semi-Markov processes. *Methodology and Computing in Applied Probability*, 6:109–127, 2004.

V. Girardin and N. Limnios. Entropy rate and maximum entropy methods for countable semi-Markov chains. *Communications in Statistics*, 33(3):609–622, 2004.

F. Gobel and A. A. Jagers. Random walks on graphs. *Stochastic Processes and their Applications*, 2:311–336, 1974.

D. Gusfield. *Algorithms on strings, trees, and sequences.* Cambridge University Press, 1997.

P. Hanna, M. Robine, and T. Rocher. An alignment based system for chord sequence retrieval. In *Proceedings of the IEEE/ACM International Joint Conference on Digital Libraries (JCDL)*, pages 101–104, 2009.

L. R. Harris. The heuristic search and the game of chess: A study of quiescence, sacrifices, and plan oriented play. In *Proceedings of the 4th International Joint Conferences on Artificial Intelligence*, pages 334–339, 1975.

M. Holmes. *Introduction to the foundations of applied mathematics.* Springer, 2002.

T. Horváth, T. Gärtner, and S. Wrobel. Cyclic pattern kernels for predictive graph mining. In *Proceedings of Knowledge Discovery and Data mining (KDD)*, pages 158–167, 2004.

X. Huang, Y. Ariki, and M. Jack. *Hidden Markov models for speech recognition.* Columbia University Press, 1990.

T. Hwa and M. Lässig. Similarity detection and localization. *Physical Review Letters*, 76(14):2591, 1996.

Y.K. Hwang and N. Ahuja. A potential field approach to path planning. *IEEE Transactions on Robotics and Automation*, 8(1):23–32, 1992.

T. Jaakkola, M. Diekhans, and D. Haussler. A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology*, 7 (1-2):95–114, 2000.

K. Jacobs. *Stochastic processes for physicists: understanding noisy systems.* Cambridge University Press, 2010.

E. T. Jaynes. Information theory and statistical mechanics. *Physical Review*, 106:620–630, 1957.

F. Jelinek. *Statistical methods for speech recognition.* The MIT Press, 1997.

G. H. John. When the best move isn't optimal: Q-learning with exploration. In *In Proceedings of the Twelfth National Conference on Artificial Intelligence*, page 1464, 1994.

K. S. Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

D. Jurafsky and J. Martin. *Speech and language processing.* Prentice-Hall, 2000.

H. J. Kappen, V. Gómez, and M. Opper. Optimal control as a graphical model inference problem. *Machine learning*, 87(2):159–182, 2012.

J. G. Kemeny and J. L. Snell. *Finite Markov Chains.* Springer-Verlag, 1976.

O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1):90–98, 1986.

D. J. Klein and M. Randic. Resistance distance. *Journal of Mathematical Chemistry*, 12:81–95, 1993.

J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *Proceedings fot the European Conference on Machine Learning*, pages 282–293, 2006.

A. Krogh, M. Brown, I. Saira Mian, K. Sjölander, and D. Haussler. Hidden markov models in computational biology: Applications to protein modeling. *Journal of Molecular Biology*, 235(5):1501–1531, 1994.

J. B. Kruskal. An overview of sequence comparison: Time warps, string edits, and macromolecules. *SIAM Review*, 25:201–237, 1983.

M. Kschischo and M. Lässig. Finite-temperature sequence alignment. *Pacific Symposium on Biocomputing*, 5:624–35, 2000.

S. M. LaValle. *Planning algorithms.* Cambridge University Press, 2006.

C. F. Lee and D. H. Wolpert. Product distribution theory for control of multi-agent systems. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, volume 2, pages 522–529. IEEE Computer Society, 2004.

C. S. Leslie, E. Eskin, and W. S. Noble. The spectrum kernel: A string kernel for SVM protein classification. In *Pacific Symposium on Biocomputing*, pages 566–575, 2002.

C. S. Leslie, E. Eskin, A. Cohen, J. Weston, and W. S. Noble. Mismatch string kernels for discriminative protein classification. *Bioinformatics*, 20 (4):467–476, 2004.

Z. X. Li and T. D. Bui. Robot path planning using fluid model. *Journal of Intelligent and Robotics Systems*, 21(1):29–50, 1998.

L. Liao and W. S. Noble. Combining pairwise sequence similarity and support vector machines for remote protein homology detection. In *Proceedings of the sixth annual international conference on Computational biology*, pages 225–232. ACM, 2002.

M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the 11th International Conference on Machine Learning (ICML)*, pages 157–163, 1994.

H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, C. Watkins, and B. Scholkopf. Text classification using string kernels. *Journal of Machine Learning Research*, 2:563–569, 2002.

C. Louste and A. Liegeois. Near optimal robust path planning for mobile robots: the viscous fluid method with friction. *Journal of Intelligent and Robotics Systems*, 27(1-2):99–112, 2000.

G. F. Luger. *Artificial intelligence: Structures ands strategies for complex problem solving, 6th ed.* Pearson International Edition, 2009.

U. V. Luxburg, A. Radl, and M. Hein. Getting lost in space: Large sample analysis of the resistance distance. In *Advances in Neural Information Processing Systems*, pages 2622–2630, 2010.

C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval.* Cambridge university press Cambridge, 2008.

A. Mantrach, L. Yen, J. Callut, K. Françoisse, M. Shimbo, and M. Saerens. The sum-over-paths covariance kernel: A novel covariance measure between nodes of a directed graph. *IEEE Transactions Pattern Analysis and Machine Intelligence*, 32(6):1112–1126, 2010.

A. Marzal and E. Vidal. Computation of normalized edit distance and applications. *IEEE Transactions Pattern Analysis and Machine Intelligence*, 15 (9):926–932, 1993.

M. Mauch, S. Dixon, C. Harte, M. Casey, and B. Fields. Discovering chord idioms through beatles and real book songs. In *Proceedings of the 8th International Society for Music Information Retrieval Conference (ISMIR)*, pages 255–258, 2007.

R. M. Mazo. *Brownian motion: fluctuations, dynamics, and applications.* Oxford University Press, 2002.

D. Michie. Game-playing and game-learning automata. *Advances in Programming and Non-Numerical Computation*, pages 183–200, 1966.

I. Millington. *Artificial intelligence for games.* Morgan Kaufmann Publishers Inc., 2006.

S. Miyazawa. A reliable sequence alignment method based on probabilities of residue correspondences. *Protein Engineering*, 10(8):999–1009, 1995.

M. F. Moens. *Information extraction: algorithms and prospects in a retrieval context.* Springer, 2006.

P. Morris. *Introduction to Game Theory.* Springer, 1994.

E. W. Myers. Ano (nd) difference algorithm and its variations. *Algorithmica*, 1(1-4):251–266, 1986.

G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.

L. A. Newberg and C. E. Lawrence. Exact calculation of distributions on integers, with application to sequence alignment. *Journal of computational biology : a journal of computational molecular cell biology*, 16(1):1–18, 2009.

A. Newell, J.C. Shaw, and H.A. Simon. Chess playing programs and the problem of complexity. *IBM Journal of Research and Development*, 4(2): 320–335, 1958.

N. J. Nilsson. *Artificial intelligence: A new synthesis.* Morgan Kaufmann Publishers Inc., 1998.

J. R. Norris. *Markov Chains.* Cambridge University Press, 1997.

J. Oncina and M. Sebban. Learning stochastic edit distance: Application in handwritten character recognition. *Pattern Recognition*, 39(9):1575–1587, 2006.

B. J. Oommen and K. Zhang. The normalized string editing problem revisited. *IEEE Trans. PAMI*, 18(6):669–672, 1996.

M. J. Osborne. *An introduction to game theory.* Oxford University Press, 2004.

F. Pachet. Surprising harmonies. In *Proceedings of the 2nd International Conference on Computing Anticipatory Systems*, pages 139–161, 1999.

L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: bringing order to the web. 1999.

H. Papadopoulos and G. Peeters. Joint estimation of chords and downbeats from an audio signal. *IEEE Transactions on Audio, Speech & Language Processing*, 19(1):138–152, 2011.

S. D. Patek and D. P. Bertsekas. Stochastic shortest path games. *SIAM Journal on Control and Optimization*, 37(3):804–824, 1997.

J. Paulus, M. Müller, and A. Klapuri. Audio-based music structure analysis. In *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR)*, pages 625–636, 2010.

J. Pickens and T. Crawford. Harmonic models for polyphonic music retrieval. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*, pages 430–437, 2002.

A. Plaat, J. Schaeffer, W. Pijls, and A. De Bruin. Best-first fixed-depth game-tree search in practice. In *Proceedings of the 14th International Joint Conferences on Artificial Intelligence*, pages 273–279, 1995.

J. Ponstein. Self-avoiding paths and the adjacency matrix of a graph. *SIAM Journal on Applied Mathematics*, 14(3):600–609, 1966.

L. R. Rabiner. Readings in speech recognition. chapter A tutorial on hidden Markov models and selected applications in speech recognition, pages 267–296. Morgan Kaufmann Publishers Inc., 1990.

L. R. Rabiner and B. H. Juang. *Fundamentals of speech recognition.* Prentice Hall PTR, 1993.

N. G. Rambidi and D. Yakovenchuck. Finding paths in a labyrinth based on reaction-diffusion media. *Biosystems*, 51(2):67 – 72, 1999.

E. Rasmusen. *Games and information: An introduction to game theory.* Blackwell Publishing, 1989.

R. C. Read and R. E. Tarjan. Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, 5(3):237–252, 1975.

A. Reinefeld, J. Schaeffer, and A. Marsland. Information acquisition in minimal window search. In *Proceedings of the 9th International Joint Conferences on Artificial Intelligence*, pages 1040–1043, 1985.

E. Ricci, T. De Bie, and N. Cristianini. Learning to align: a statistical approach. In *Proceedings of the 7th International Symposium on Intelligent Data Analysis (IDA)*, pages 25–36, 2007.

E. Ristad and P. Yianilos. Learning string-edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):522–532, 1998a.

E. S. Ristad and P. N. Yianilos. Learning string-edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):522–532, 1998b.

H. Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535, 1952.

S. E. Robertson. The probability ranking principle in IR. *Journal of documentation*, 33(4):294–304, 1977.

S. E. Robertson and K. S. Jones. Relevance weighting of search terms. *Journal of the American Society for Information science*, 27(3):129–146, 1976.

J. Rousu and J. Shawe-Taylor. Efficient computation of gapped substring kernels on large alphabets. *Journal of Machine Learning Research*, 6:1323–1344, 2005.

A. Rubinstein. *Modeling bounded rationality.* MIT Press, 1998.

J. Rudnick and G. Gaspari. *Elements of the random walk.* Cambridge University Press, 2004.

S. Russell and P. Norvig. *Artificial intelligence: A modern approach, 2nd Ed.* Prentice-Hall, 2003.

S. Russell and E. Wefald. *Do the right thing : studies in limited rationality.* MIT Press, 1991.

M. Saerens, Y. Achbany, F. Fouss, and L. Yen. Randomized shortest-path problems: Two related models. *Neural Computation*, 21(8):2363–2404, 2009.

R. A. Sahner, K. Trivedi, and A. Puliafito. *Performance and reliability analysis of computer systems: an example-based approach using the SHARPE software package.* Springer Science & Business Media, 2012.

H. Saigo, J. P. Vert, N. Ueda, and T. Akutsu. Protein homology detection using string alignment kernels. *Bioinformatics*, 20(11):1682–1689, 2004.

G. Salton and C. S. Yang. On the specification of term values in automatic indexing. *Journal of documentation*, 29(4):351–372, 1973.

G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

D. Sankoff and J. B. Kruskal. *Time warps, string edits, and macromolecules.* Addison-Wesley, 1983.

G. Schmidt and W. Neubauer. High-speed robot path planning in time-varying environment employing a diffusion equation strategy. In *Robotic systems: advanced techniques and applications*, pages 207–215, 1992.

G.K. Schmidt and K. Azarm. Mobile robot navigation in a dynamic world using an unsteady diffusion equation strategy. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 642–647, 1992.

B. Schölkopf and A. J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond.* MIT press, 2002.

L. Shapley. Stochastic games. *Proceedings of the National Academy of Sciences of the United States of America*, 39(10):1095–1100, 1953.

J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis.* Cambridge University Press, 2004.

N. Shervashidze and K. M. Borgwardt. Fast subtree kernels on graphs. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1660–1668, 2009.

Herbert A. Simon. Bounded rationality and organizational learning. *Organization Science*, 2(1):125–134, 1991.

S. Singh, T. Jaakkola, M. L. Littman, and C. Szepesvári. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 39(3):287–308, 2000.

J. Smed. *Algorithms and Networking for Computer.* John Wiley & Sons, 2006.

R. Somla. New algorithms for solving simple stochastic games. *Electronic Notes in Theoretical Computer Science*, 119(1):51–65, 2005.

M. Steel and J. Hein. Applying the thorne-kishino-felsenstein model to sequence evolution on a star-shaped tree. *Applied Mathematics Letters*, 14(6): 679–684, 2001.

L. Steels. Exploiting analogical representations. *Robotics and Autonomous Systems*, 6(1-2):71–88, 1990.

M. Stenmark. Synthesizing board evaluation functions for connect4 using machine learning techniques. Master's thesis, Department of Computer Science, Østfold University College, 2005.

G. Stephen. *String searching algorithms.* World Scientific, 1994.

N. R. Sturtevant and R. E. Korf. On pruning techniques for multi-player games. In *Proceedings of the 17th National Conference on Artificial Intelligence*, pages 201–207, 2000.

J. Sun, S. Boyd, L. Xiao, and P. Diaconis. The fastest mixing Markov process on a graph and a connection to a maximum variance unfolding problem. *SIAM Review*, 48(4):681–699, 2006.

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction.* The MIT Press, 1998.

A. Tahbaz and A. Jadbabaie. A one-parameter family of distributed consensus algorithms with boundary: From shortest paths to mean hitting times. In

Proceedings of IEEE Conference on Decision and Control, pages 4664–4669, 2006.

L. Tarassenko and A. Blake. Analogue computation of collision-free paths. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 540–545, 1991.

H. M. Taylor and S. Karlin. *An Introduction to Stochastic Modeling, 3th Ed.* Academic Press, 1998.

S. Theodoridis and K. Koutroumbas. *Pattern recognition, 3th edition.* Academic Press, 2006.

J. L. Thorne, H. Kishino, and J. Felsenstein. An evolutionary model for maximum likelihood alignment of dna sequences. *Journal of Molecular Evolution*, 33(2):114–124, 1991.

J. L. Thorne, H. Kishino, and J. Felsenstein. Inching toward reality: an improved likelihood model of sequence evolution. *Journal of Molecular Evolution*, 34(1):3–16, 1992.

S. Thrun. Exploration in active learning. *The handbook of brain theory and neural networks*, pages 381–384, 1998.

E. Todorov. Linearly-solvable markov decision problems. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1369–1375, 2006.

J. A. Tomlin. A new paradigm for ranking pages on the world wide web. In *Proceedings of the Twelfth International World Wide Web Conference*, pages 350–355, 2003.

H. R. Turtle and W. B. Croft. A comparison of text retrieval models. *The computer journal*, 35(3):279–290, 1992.

E. Vidal, A. Marzal, and P. Aibar. Fast computation of normalized edit distances. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(9):899–902, 1995.

S. V. N. Vishwanathan and Alexander J. Smola. Fast kernels for string and tree matching. In *Advances in Neural Information Processing Systems 15*, pages 569–576, 2003.

S. V. N. Vishwanathan, K. M. Borgwardt, and N. N. Schraudolph. Fast computation of graph kernels. In *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference*. MIT Press, Cambridge MA, USA, 2007.

S. V. N. Vishwanathan, K. M. Borgwardt, R. I. Kondor, and N. N. Schraudolph. Graph kernels. *Journal of Machine Learning Research*, 11:1201–1242, 2010.

P. M. B. Vitányi. Information distance in multiples. *IEEE Transactions on Information Theory*, 57(4):2451–2456, 2011.

A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, 1967.

R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.

S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.

M. S. Waterman and M. Eggert. A new algorithm for best subsequence alignments with application to tRNA-rRNA comparisons. *Journal of Molecular Biology*, 197(4):723 – 728, 1987.

C. Watkins. Kernels from matching operations. Technical report, Department of Computer Science, Royal Holloway, University of London, 1999.

A. Weigel and F. Fein. Normalizing the weighted edit distance. In *Proceedings of the International Conference on Pattern Recognition*, pages 399–402, 1994.

D. Wolpert. Information theory – the bridge connecting bounded rational game theory and statistical physics. In *Complex Engineered Systems*, volume 14, pages 262–290. Springer Berlin, 2006.

L. Yen, A. Mantrach, M. Shimbo, and M. Saerens. A family of dissimilarity measures between nodes generalizing both the shortest-path and the commute-time distances. In *Proceedings of the 14th International Conference on Knowledge Discovery and Data Mining*, pages 785–793, 2008.

L. Yujian and L. Bo. A normalized levenshtein distance metric. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1091–1095, 2007.

M. Q. Zhang and T. G. Marr. Alignment of molecular sequences seen as random path analysis. *Journal of Theoretical Biology*, 174(2):119–29, 1995.