CPT+ : UN MODÈLE COMPACT POUR LA PRÉDICTION DE SÉQUENCES DE SYMBOLES
AVEC HAUTE EXACTITUDE

THÈSE PRÉSENTÉE À LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET DE LA
RECHERCHE EN VUE DE L'OBTENTION DE LA MAÎTRISE ÈS SCIENCES EN
INFORMATIQUE

TED GUENICHE

DÉPARTEMENT D’INFORMATIQUE
FACULTÉ DES SCIENCES
UNIVERSITÉ DE MONCTON

Mai 2015
COMPOSITION DU JURY

Président du jury :

Julien Chiasson
Professeur d’informatique, Université de Moncton

Examinateur externe :

Engelbert Mephu Nguifo
Professeur d’informatique, Université Blaise Pascal Clermont-Ferrand

Examinateur interne :

Éric Hervet
Professeur d’informatique, Université de Moncton

Directeur de thèse :

Philippe Fournier-Viger
Professeur d’informatique, Université de Moncton
REMERCIEMENTS

Je tiens à remercier mon directeur de thèse, Philippe Fournier-Viger, qui m’a fait découvrir la fouille de données et m’a introduit à la recherche. Philippe a été mon mentor au fil de ces dernières années ; il m’a guidé et encouragé à travers de nombreux problèmes tous aussi stimulants les uns que les autres. Merci Philippe de m’avoir motivé à pousser ma recherche.

Je tiens également à adresser mes remerciements aux membres de mon jury, qui ont accepté d’évaluer mon travail : Engelbert Mephu Nguifo, Éric Hervet et Philippe Fournier-Viger.

Enfin, je tiens à exprimer ma gratitude à mes parents et à Emilie pour leur support et leurs encouragements tout au long de ma maîtrise. Ces trois années n’auraient pas été possibles sans leurs soutiens.
# TABLE DES MATIÈRES

RÉSUMÉ  ........................................................................................................... xi
RÉSUMÉ (anglais) .......................................................................................... xii
INTRODUCTION GÉNÉRALE ....................................................................... 1

CHAPITRE I
COMPACT PREDICTION TREE : UN MODÈLE SANS PERTE POUR LA PRÉDICATION DE SÉQUENCES .................................................. 6
1 Introduction ................................................................................................. 7
2 Travaux antérieurs ...................................................................................... 8
3 Compact Prediction Tree ............................................................................ 10
   3.1 Entraînement ...................................................................................... 10
   3.2 Prédiction ......................................................................................... 12
   3.3 Optimisations .................................................................................... 14
4 Résultats expérimentaux .......................................................................... 14
   4.1 Jeux de données .............................................................................. 15
   4.2 Environnement d’évaluation ............................................................. 16
   4.3 Épures ..............................................................

5 Conclusion .................................................................................................. 23

CHAPITRE II
RÉDUCTION DE LA COMPLEXITÉ SPATIALE ET TEMPORELLE DU COMPACT PREDICTION TREE POUR LA PRÉDICATION DE SÉQUENCES .................................................. 24
1 Introduction ................................................................................................. 25
2 Compact Prediction Tree ............................................................................ 27
3 Stratégies de compression de l’arbre de prédiction .................................. 29
   3.1 Compression des Chaînes Fréquentes (FSC) ................................. 29
   3.2 Compression des Branches Simples (SBC). ................................. 30
4 Stratégie de réduction des temps de prédiction ...................................... 31
5 Évaluation expérimentale ........................................................................ 32
6 Conclusion .................................................................................................. 38

CONCLUSION GÉNÉRALE ........................................................................... 39

ANNEXE A
WBPL : UNE LIBRAIRIE OPEN SOURCE POUR LA PRÉDICATION DE COMPORTEMENT SUR LE WEB ................................................................. 42
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
<td>42</td>
</tr>
<tr>
<td>2</td>
<td>Modèles utilisés</td>
<td>43</td>
</tr>
<tr>
<td>3</td>
<td>Interface graphique</td>
<td>45</td>
</tr>
<tr>
<td>4</td>
<td>Utilisation de la librairie</td>
<td>47</td>
</tr>
<tr>
<td>5</td>
<td>Conclusion</td>
<td>48</td>
</tr>
</tbody>
</table>

RÉFÉRENCES 50
# TABLE OF CONTENTS

**ABSTRACT** (French) ................................................................. xi

**ABSTRACT** .................................................................................. xii

**GENERAL INTRODUCTION** .......................................................... 1

**CHAPTER I**
**COMPACT PREDICTION TREE: A LOSSLESS MODEL FOR ACCURATE SEQUENCE PREDICTION** ................................................................. 6

1 Introduction .................................................................................. 7

2 Related Work .............................................................................. 8

3 Compact Prediction Tree ................................................................. 10

3.1 Training .................................................................................. 10

3.2 Prediction .............................................................................. 12

3.3 Optimizations ......................................................................... 14

4 Experimental Evaluation ................................................................. 14

4.1 Datasets ................................................................................ 15

4.2 Evaluation Framework .............................................................. 16

4.3 Experiments ........................................................................... 18

5 Conclusion ................................................................................ 23

**CHAPTER II**
**CPT+: DECREASING THE TIME/SPACE COMPLEXITY OF THE COMPACT PREDICTION TREE** ................................................................. 24

1 Introduction ................................................................................ 25

2 Compact Prediction Tree ................................................................. 27

3 Compression strategies ................................................................. 29

3.1 Frequent subsequence compression (FSC) ..................................... 29

3.2 Simple Branches Compression (SBC) .......................................... 30

4 Time reduction strategy ................................................................. 31

5 Experimental evaluation ............................................................... 32

6 Conclusion ................................................................................ 38

**GENERAL CONCLUSION** ............................................................ 39

**APPENDIX A**
**WBPL : AN OPEN-SOURCE LIBRARY FOR PREDICTING WEB SURFING BEHAVIORS** ................................................................. 42
| 1  | Introduction ................................................................. | 42 |
| 2  | Presented Models ............................................................ | 43 |
| 3  | Using the Web GUI ............................................................ | 45 |
| 4  | Using the Library ............................................................. | 47 |
| 5  | Conclusion ................................................................. | 48 |
|    | REFERENCES .............................................................. | 50 |

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-1</td>
<td>Dataset characteristics</td>
<td>16</td>
</tr>
<tr>
<td>I-2</td>
<td>Experiment’s parameters</td>
<td>18</td>
</tr>
<tr>
<td>I-3</td>
<td>Parameters of CPT</td>
<td>18</td>
</tr>
<tr>
<td>I-4</td>
<td>Comparison of accuracy and model size</td>
<td>19</td>
</tr>
<tr>
<td>I-5</td>
<td>Comparison of training time and testing time</td>
<td>19</td>
</tr>
<tr>
<td>II-1</td>
<td>Datasets</td>
<td>34</td>
</tr>
<tr>
<td>II-2</td>
<td>Prediction models and their accuracy</td>
<td>38</td>
</tr>
</tbody>
</table>
LISTE DES FIGURES

I-1 DG, PPM and AKOM trained with training sequences $s_1 = \{A, B, C, A, C, B, D\}$ and $s_2 = \{C, C, A, B, C, B, C, A\}$. ................................................................. 9
I-2 Prediction Tree (PT), Inverted Index (II) and Lookup Table (LT). .................. 12
I-3 Sequence splitting (context, prefix, suffix) ...................................................... 17
I-4 Comparison of scalability ................................................................. 20
I-5 Influence of prefix length on accuracy and model size ................................ 21
I-6 Influence of the Recursive Divider optimization ........................................... 22
I-7 Influence of the Sequence Splitter optimization ........................................... 22
II-1 Building CPT structures ................................................................. 28
II-2 Application of the FSC and SBC compression strategies ............................... 31
II-3 Compression rate and training time of the compression strategies ................. 35
II-4 Execution time and accuracy gains provided by the PNR strategy ................. 36
II-5 Scalability of the prediction models. ...................................................... 37
A-1 WSBL user interface ................................................................. 45
A-2 Results section ................................................................. 48
**LISTE DES ALGORITHMES**

1. The prediction process of CPT .......................... 13
2. Recursive divider strategy ............................. 15
3. The prediction algorithm using PNR ................. 33
ACRONYMES

AKOM  All-k-order Markov
CPT   Compact Prediction Tree
DG    Dependency Graph
FSC   Frequent Subsequence Compression
GUI   Graphical User Interface
II    Inverted Index
LT    Lookup Table
PNR   Prediction with improved Noise Reduction
PPM   Prediction by Partial Matching
PT    Prediction Tree
RC    Recursive Divider
SBC   Simple Branches Compression
TDAG  Transition Directed Acyclic Graph
VMM   Variable-order Markov model
RÉSUMÉ

La prédiction de séquence consiste à prédire les prochains symboles (événements) d’une séquence sur la base d’un ensemble de séquences d’entraînement. Cette tâche a de nombreuses applications telles que le préchargement de pages Web, la recommandation de produits, la prévision météorologique et la prédiction des tendances du marché boursier. Plusieurs modèles de prédiction de séquences ont été proposés, la plupart étant des variations des Variable order Markov Models, par exemple PPM, DG, CTW, TDAG et All-K-order Markov (AKOM). Toutefois, ils souffrent de deux limites importantes. Premièrement, ils partent de l’hypothèse Markovienne qu’un événement ne dépend que de son prédécesseur. Or, cette hypothèse ne tient pas pour plusieurs applications. Deuxièmement, ce sont des modèles construits avec perte d’information par rapport aux séquences d’entraînement.

Pour repousser ces limites, nous proposons un modèle de prédiction nommé Compact Prediction Tree (CPT). Il compresse les séquences d’entraînement afin que toute l’information pertinente soit disponible pour chaque prédiction, et emploie un mécanisme d’indexation pour permettre un accès rapide aux séquences. De plus, une mesure de similarité est proposée pour tenir compte de plusieurs événements antérieurs lors des prédictions et tolérer des variations d’ordre dans les séquences. Une comparaison expérimentale de CPT avec PPM, AKOM et DG sur six jeux de données réels montre que CPT obtient une meilleure exactitude pour cinq d’entre eux avec un gain atteignant jusqu’à 12%, tout en ayant une faible complexité temporelle pour la phase d’entraînement. Néanmoins, la complexité spatiale de CPT est bien plus importante que celle de DG et PPM, bien qu’elle soit plus petite que celle de AKOM et TDAG. De plus, CPT a une forte complexité temporelle pour la prédiction. Pour remédier à ces problèmes, nous introduisons une version améliorée nommée CPT+, qui incorpore deux stratégies de compression pour réduire la taille du modèle et une troisième stratégie pour accélérer son processus de prédiction. Des résultats expérimentaux montrent que l’arbre de CPT+ est jusqu’à 98 fois plus petit que celui de CPT et que CPT+ fait des prédictions jusqu’à 4,5 fois plus rapide.

Mots clés: prédiction de séquence ; modèles de prédiction ; compression
ABSTRACT

Sequence prediction consists in predicting the next item(s) of a sequence of items, given a set of training sequences and a finite alphabet of items (symbols). This task has numerous applications such as web page prefetching, product recommendation, weather forecasting and stock market prediction. Many sequence predictions models have been proposed. Some of the most popular models are variations of Variable order Markov Models such as PPM, DG, CTW, TDAG and All-K-order Markov (AKOM). However, these predictions models suffer from two important limitations. First, they are based on the Markov property, which states that the occurrence of an item in a sequence depends only on the preceding item. While this assumption holds in some applications, for others it does not, and thus prediction accuracy using these models can severely decrease. Second, these models are lossy models, i.e. the original training sequences cannot be recovered from these models.

To overcome these problems, we introduce a novel sequence prediction model named the Compact Prediction Tree (CPT). CPT is built by losslessly compressing the training sequences to ensure that all relevant information is available for each prediction. Furthermore, it provides an indexing mechanism to allow fast sequence searching and matching. When performing predictions, CPT also relies on a novel sequence similarity measure to assess if non-identical sequences are nonetheless similar. The similarity measure is dynamically loosened by CPT to tolerate sequence variations. An experimental study comparing CPT with PPM, AKOM and DG on six real datasets shows that CPT is more accurate for five of them, offering up to 12% higher accuracy while also having a low time complexity for training. However, CPT consumes much more memory than the PPM and DG models, but requires less than AKOM and TDAG. CPT’s execution time for prediction is also quite higher than the other presented models. To overcome these issues we introduce an improved version of CPT named CPT+. CPT+ proposes two compression strategies to reduce its size and a third strategy to enhance its prediction time. Our experiments have shown that CPT+ is up to 98 times smaller than CPT and CPT’s prediction time is up to 4.5 times less.

Keywords: sequence prediction; predictive models; compression
La prédiction de séquences est une tâche fondamentale en fouille de données [4] qui consiste à prédire le(s) prochain(s) symbole(s) d’une séquence de symboles. Une application classique de la prédiction de séquences est la compression sans perte [13], mais il existe aussi de nombreuses applications dans des domaines variés telles que le préchargement de pages Web [2, 9, 18], les prévisions météorologiques [10], la prédiction des tendances du marché boursier [11], l’analyse de séquences de protéines [8] et la recommandation de produits de consommation sur le Web [9].

Les systèmes de prédiction employant la prédiction de séquences sont généralement spécialisés pour une ou plusieurs applications [9, 27]. Ces systèmes supposent aussi souvent que de l’information supplémentaire accompagne les séquences pour en indiquer le contexte. L’utilisation d’information contextuelle a pour but d’augmenter l’exactitude des prédictions. Dans le contexte de prédiction de produits, chaque utilisateur peut, par exemple, se voir assigner un profil contenant sa région géographique et ses intérêts. Les éléments (symboles) qui constituent les séquences peuvent eux aussi être annotés avec des données supplémentaires telles qu’une date, une quantité ou un prix.

Dans le cadre de ce travail, nous nous intéressons au problème de prédiction de séquences, défini de la façon suivante [4]. Soit un alphabet $Z = \{e_1, e_2, \ldots, e_m\}$ contenant un ensemble d’éléments (symboles). Une séquence est une suite d’éléments totalement ordonnée $s = \langle i_1, i_2, \ldots i_n \rangle$, où $i_k \in Z$ ($1 \leq k \leq n$). Un modèle de prédiction $M$ est entrainé avec un ensemble de séquences d’entraînement. Une fois entraîné, le modèle peut être utilisé pour effectuer des prédictions. Une prédiction consiste à prédire le prochain élément $i_{n+1}$ d’une séquence $\langle i_1, i_2, \ldots i_n \rangle$ en utilisant le modèle $M$.

De nombreux modèles de prédiction de séquences ont été proposés [2, 14, 18, 26]. Un grand nombre d’entre eux sont des variations des Variable-order Markov models (VOM ou VMM) [2, 14, 18]. Ces modèles utilisent les données d’entraînement pour créer un ou plusieurs automates à états finis. Les états initiaux de ces automates correspondent chacun à un suffixe et chaque transition indique la probabilité de passer à un état final qui représente une prédiction. Le suffixe d’une séquence
$s = \langle i_1, i_2, ... i_n \rangle$ dénoté $P_y(s)$ est défini comme étant les $y$ derniers éléments de la séquence, c’est-à-dire $P_y(s) = \langle i_{n-y+1}, i_{n-y+2} ... i_n \rangle$ (où $y \geq 1$). La longueur $y$ des suffixes utilisée pour créer les automates est appelée l’*ordre* du modèle et peut être variable. Des variations populaires des VMM sont *Prediction by Partial Matching* (PPM) [1], *Dependency Graph* (DG) [2], *All-k-order-Markov* (AKOM) [3], *Transition Directed Acyclic Graph* (TDAG) [4] et *Probabilistic Suffix Trees* (PST) [5].

D’autre part, un certain nombre d’algorithmes de compression ont été adaptés pour la prédiction de séquences tels que *Context Tree Weighting* (CTW) [6], *Active Lezi* [7] et *LZ78* [12], utilisé notamment dans le format d’image PNG. La compression sans perte est fortement liée à la prédiction de séquences sur un alphabet fini et peut être considérée comme une de ses applications [13]. En effet, le processus de compression (encodage) est équivalent au processus d’entraînement de la prédiction de séquences. La différence principale est dans l’utilisation du modèle construit.

Les modèles basés sur les VMM ont plusieurs limitations : premièrement, ils utilisent seulement une partie des données d’entraînement ; seulement des chaînes ordonnées d’au plus $x$ éléments contigus sont utilisés pour la création du modèle et considérées lors de la prédiction, où $x$ est l’ordre du modèle. Deuxièmement, lorsque l’ordre du modèle est grand, la complexité spatiale devient alors très importante et peut rendre le modèle inutilisable. Bien que des propositions aient été faites pour réduire la complexité temporelle et spatiale de ces modèles [14], l’exactitude de leurs prédictions a subi peu d’amélioration.

Une autre limite importante des VMM est la faible capacité à tolérer le bruit [4, 14]. La tolérance au bruit d’un modèle est définie comme la capacité à reconnaître que deux séquences sont partiellement similaires en tenant compte de variations dans l’ordre des éléments ou de l’absence de certains éléments dans les séquences. Un modèle qui ne tolère pas le bruit considère que toutes paires de séquences sont soit identiques, soit totalement différentes. Considérons, une séquence à prédire $s = \langle i_1, i_2, ... i_n \rangle$ sur la base d’un ensemble de séquences d’entraînement. Si aucune séquence d’entraînement ne possède un suffixe identique à un suffixe de $s$, alors les modèles basés sur les VMM ne peuvent pas faire de prédiction pour $s$. La tolérance au bruit a donc un impact direct sur le nombre de séquences qu’un modèle est capable de prédire. Certains modèles tels que DG et TDAG sont plus flexibles dans le sens où ils permettent d’identifier des sous-séquences similaires à un suffixe de $s$ mais ces mécanismes ont soit un effet négatif sur l’exactitude ou ont un coût spatial.
Plusieurs approches provenant d’autres domaines ont aussi été adaptées pour la prédiction de séquences, notamment les règles séquentielles [9, 20] et les réseaux de neurones récurrents [26, 27]. Ces approches sont aussi limitées par la perte d’information lors de la construction des modèles. De plus les règles séquentielles ont une très grande complexité spatiale.

**Objectif.** L’objectif de ce travail est de proposer un nouveau modèle palliant ces limites afin d’offrir des prédictions avec une exactitude plus élevée et une importante tolérance au bruit.

**Hypothèse.** Les deux principales hypothèses de ce travail sont :

— Un modèle de prédiction compressant les séquences d’entraînement pourra utiliser toute l’information pertinente pour chaque prédiction, ce qui permettra d’atteindre une exactitude plus élevée.

— Utiliser une nouvelle mesure de similarité tenant compte de plusieurs éléments antérieurs pour la prédiction, des variations d’ordre et de l’absence d’éléments permettra d’augmenter l’exactitude du modèle et sa tolérance au bruit.

**Contribution.** La contribution de cette thèse est la proposition d’un nouveau modèle de prédiction nommé Compact Prediction Tree (CPT), un modèle tolérant au bruit et sans perte qui est rapide et a une haute exactitude. CPT est un modèle qui compresse les séquences d’entraînement dans un arbre préfixé. Contrairement aux modèles basés sur l’hypothèse de Markov, CPT est construit avec les séquences entières du jeu de données d’entraînement. Cette représentation permet l’utilisation de séquences complètes pour la tâche de prédiction tout en assurant une complexité spatiale acceptable. À son état le plus pur, sans optimisation, CPT est un algorithme de compression sans perte où l’intégralité des données d’entraînement peut être entièrement ré générée. La tolérance au bruit de CPT est due à son processus de prédiction qui se base sur une nouvelle mesure de similarité pour identifier des séquences similaires lors d’une prédiction. De plus CPT dispose d’un mécanisme capable de relâcher dynamiquement le seuil nécessaire pour considérer que deux séquences sont similaires, afin d’assurer qu’un nombre minimal de séquences similaires soient identifiées pour toute prédiction. Ce relâchement permet à CPT de faire des prédictions exactes tout en tenant compte du bruit.
**Organisation de la thèse.** La thèse est une thèse par publications. Les chapitres 2 et 3 correspondent à deux publications parues dans des actes de conférences internationales en fouille de données :


— La seconde publication, « CPT+ : Decreasing the time/space complexity of the Compact Prediction Tree », a été acceptée pour publication dans les actes de *PAKDD 2015 : The 19th Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Cette deuxième publication introduit trois stratégies ; deux pour réduire la taille de l’arbre de prédiction de CPT et une pour réduire les temps de calcul lors du processus de prédiction. La publication présente aussi une évaluation des gains spatiaux introduits par les stratégies de compression et une comparaison de CPT+ avec un plus grand nombre de modèles et de jeux de données qu’au chapitre 1.

Il est à noter que les publications ont été légèrement modifiées pour ajouter quelques détails omis par manque d’espace dans les versions publiées.

Finalement, le chapitre 4 présente la conclusion générale de la thèse et les perspectives de travaux futurs.

Par ailleurs, une troisième publication est en annexe de la thèse. Elle est intitulée « WBPL : An Open-Source Library for Predicting Web Surfing Behaviors ». Elle introduit WBPL (*Web users Behavior Prediction Library*), une librairie à code ouvert que nous avons conçue pour l’utilisation de modèles de prédiction de séquences dans des applications web. Elle permet de comparer et d’utiliser les modèles de prédiction CPT, CPT+ ainsi que les autres modèles comparés dans cette thèse, dans des applications web de recommandation. Cette publication est parue dans les actes
de la conférence *ISMIS 2014 : The 21\textsuperscript{th} International Symposium on Methodologies for Intelligent Systems*. 
CHAPITRE I

COMPACT PREDICTION TREE: A LOSSLESS MODEL FOR ACCURATE SEQUENCE PREDICTION

COMPACT PREDICTION TREE: UN MODÈLE SANS PERTE POUR LA PRÉDICTION DE SÉQUENCES
Abstract

Predicting the next item of a sequence over a finite alphabet has important applications in many domains. In this paper, we present a novel prediction model named CPT (Compact Prediction Tree) which losslessly compresses the training data so that all relevant information is available for each prediction. Our approach is incremental, offers a low time complexity for its training phase and is easily adaptable for different applications and contexts. We compared the performance of CPT with state of the art techniques, namely PPM (Prediction by Partial Matching), DG (Dependency Graph) and All-$K$-th-Order Markov. Results show that CPT yields higher accuracy on most datasets (up to 12% more than the second best approach), has better training time than DG and PPM, and is considerably smaller than All-$K$-th-Order Markov.

Keywords: sequence prediction, next item prediction, accuracy, compression

1 Introduction

Given a set of training sequences, the problem of sequence prediction consists in finding the next element of a target sequence by only observing its previous items. The number of applications associated with this problem is extensive. It includes applications such as web page prefetching [2, 18], consumer product recommendation [9], weather forecasting [10] and stock market prediction [11].

The literature on this subject is extensive and there are many different approaches [19]. Two of the most popular are PPM (Prediction by Partial Matching) [1] and DG (Dependency Graph) [2]. Over the years, these models have been greatly improved in terms of time and memory efficiency [3,18], but their performance remains more or less the same in terms of prediction accuracy. Markov Chains are also widely used for sequence prediction. However, they assume that sequences are Markovian. Other approaches exist such as neural networks and association rules [20, 26, 27], but all these approaches build prediction lossy models from training sequences. Therefore, they do not use all the information available in training sequences for making predictions.

In this paper, we propose a novel approach for sequence prediction that uses the whole infor-
information from training sequences to perform predictions. The hypothesis is that it would increase prediction accuracy. There are however several important challenges to build such an approach. First, it requires a structure for storing the whole information efficiently in terms of storage space. Second, the structure should be efficiently updatable if new sequences are added. Third, it is necessary to define an algorithm for performing predictions using the data structure that is time efficient and generate accurate predictions.

We address all these challenges. First, we propose an efficient trie-based data structure named CPT (Compact Prediction Tree) which losslessly compresses all training sequences. The construction process of the CPT structure is incremental, offers a low time complexity and is reversible (i.e. it is possible to restore the original dataset from a CPT). Second, we propose an efficient algorithm to perform sequence predictions using the CPT structure. Thanks to CPT’s indexing mechanism, the algorithm can quickly collect relevant information for making a prediction. Third, we introduce two strategies that respectively reduce the size of CPT and increase prediction accuracy. Lastly, we perform an extensive experimental study to compare the performance of our approach with state of the art sequence prediction algorithms, namely PPM [1](Prediction by Partial Matching), DG [2] (Dependency Graph) and All-Kth-Order Markov [3], on several real-life datasets. Results show that CPT yield superior accuracy in most cases.

This paper is organized as follows. In section 2, we formally present the prediction problem and discuss related work. In section 3, we present CPT, explain how its substructures are built and how it is used to perform predictions. In section 4, we describe an experimental study. Finally, in section 5, we present our conclusions.

2 Related Work

Given a finite alphabet \( I = \{i_1, i_2, ..., i_m\} \), an individual sequence of items is defined as \( S = \langle s_1, s_2, ..., s_n \rangle \), a list of ordered items where \( s_i \in I \ (1 \leq i \leq m) \). Let \( T = \{S_1, S_2, ..., S_t\} \) be a set of training sequences used to build a prediction model \( M \). The problem of sequence prediction consists in predicting the next item \( s_{n+1} \) of a given sequence \( \langle s_1, s_2, ..., s_n \rangle \) by using the prediction model \( M \) based on our previous knowledge built from \( T \).
In this section we describe three sequence prediction models: PPM, DG, and AKOM. These models are popular in the literature [3, 18, 19] and are all variations of Variable-order Markov models (VOM or VMM). In fig. I-1 we depict the resulting data structure for all three models with training sequences \( S_1 = \{A, B, C, A, C, B, D\} \) and \( S_2 = \{C, C, A, B, C, B, C, A\} \).

A PPM model of order \( K \) predicts the next item of a sequence based on the last \( K \) items of the sequence. A PPM model can be represented as a graph where prefix subsequences are linked to suffix subsequences by outgoing arcs having transition probabilities. For example, in Fig. I-1, a PPM model of order 1 is illustrated. The leftmost subtree of this model indicates that the conditional probabilities \( P(B|A) \) and \( P(C|A) \) are estimated to be respectively \( 2/4 \) and \( 1/4 \). To predict the next item of a sequence using a \( K \)-Order PPM, the last \( k \) items of the sequence are matched with one of the node and then the transition with the highest value is chosen as the prediction. This approach yields good results for certain applications [1, 18]. However, an important drawback is
its rigidity toward the patterns that it can learn. The smallest variation in a subsequence will affect the prediction outcome, and thus prediction accuracy. This problem becomes worse for noisy datasets. In a K-Order PPM model only the K-th-order Markov predictor is used. In the All-K-Order Markov Model [3], all Markov predictors from 1 to K inclusively are used. This has the advantage of yielding higher accuracy in most cases [18], but it suffers from a much higher state and space complexity. A lot of research has been done to improve the speed and memory requirements of these approaches, for example by pruning states [3, 18, 19]. In fig. I-1, AKOM has an order of 2.

The Dependency Graph (DG) [2] model is a graph where each node represents an item \( i \in I \). A directional arc connects a node \( A \) to a node \( B \) if and only if \( B \) appears within \( x \) items from \( A \) in training sequences, where \( x \) is a user-defined parameter named the lookahead window length. The weight of the arc is defined as \( P(B|A)/P(A) \). In fig. I-1, DG has a lookahead window length of 2.

There are many other approaches to sequence prediction such as using sequential rules [20], neural networks and Context Tree Weighting [6] (see [26] for an overview). However, all these approaches build lossy models, which may thus ignore relevant information from training sequences when making predictions. In this work, we propose a lossless prediction model. Our hypothesis is that using all the relevant information from training sequences to make predictions would increase prediction accuracy.

3 Compact Prediction Tree

In this section, we present our approach. It consists of two phases: training and prediction.

3.1 Training

In the training phase, our prediction model named the Compact Prediction Tree is built. It is composed of three data structures: (1) a Prediction Tree (PT), (2) an Inverted Index (II) and (3) a Lookup Table (LT). The training is done using a training dataset composed of a set of sequences. Sequences are inserted one at a time in the PT and the II. For example, Figure I-2 shows the PT, II and LT constructed from sequences \( \langle A, B, C \rangle, \langle A, B \rangle, \langle A, B, D \rangle, \langle B, C \rangle \) and \( \langle B, D, E \rangle \).
The *Prediction Tree* is recursively defined as a node. A node contains an item, a list of children nodes and a pointer to its parent node. A sequence is represented within the tree as a full branch or a partial branch, starting from a direct child of the root node. The prediction tree is constructed as follows: given a training sequence, we check if the current node (the root) has a direct child matching the first item of this sequence. If it does not, a new child is inserted to the root node with this item’s value. Then, the cursor is moved to the newly created child and this process is repeated for the next item in the training sequence. The construction of this tree for \( N \) training sequences takes \( O(N) \) in time and is done by reading the sequences one by one with a single pass over the data. The space complexity of the PT is in the worst case \( O(N \times \text{averageLengthOfSequences}) \), but in the average case the PT is more compact because the branches often overlap by sharing nodes. Two sequences share their first \( v \) nodes in the PT if they share a prefix of \( v \) items. The PT is incrementally updatable and is fast to construct.

The second structure is the *Inverted Index*. It is designed to quickly find in which sequences a given item appears. Hence, it can also be used to find all the sequences containing a set of items. The II is defined as a hash table containing a key for each unique item encountered during the training. Each key leads to a bitset that indicates IDs of the sequences where the item appears. A bitset contains \( n \) bits, where \( n \) is the number of training sequences. The presence of an item in the \( s \)-th sequence is indicated by setting the \( s \)-th bit to 1 in its bitset, and 0 otherwise. The II, just like the PT, has an average construction time of \( O(n) \) and takes \( (n + b) \times u \) bits where \( n \) is the number of training sequences, \( u \) is the number of unique items, and \( b \) is the size of an item in bits.

The third and last structure is the *Lookup Table*. It links the II to the PT. For each sequence ID, the LT points to the last node of the sequence in the PT. The LT purpose is to provide an efficient way to retrieve sequences from the PT using their sequence IDs. The LT is updated after each sequence insertion in the PT. Its time complexity is \( O(n) \) where \( n \) is the number of sequences. In terms of size, this data structure takes \( n \times (b + p) \) bytes where \( n \) is the number of sequences, \( b \) is the size of an item in bytes, and \( p \) is the size of a pointer in bytes. The addition of the LT to the PT makes it a lossless representation of the training set of sequences, i.e. it allows restoring the original dataset.
The training process is really fast ($O(n)$). The CPT takes more or less space depending on the dataset. If many sequences share common prefixes, a greater compression is achieved. Note that the PT itself could be further compressed by replacing frequent subsequences by single nodes or pruning infrequent nodes. These optimizations are outside the scope of this paper and will be investigated in future work.

3.2 Prediction

In the prediction phase (Algorithm 3), our prediction model is used to perform predictions. Let $x$ be an integer named the prefix length. Making a prediction for a given sequence $S$ is done by finding all sequences that contain the last $x$ items from $S$ in any order and in any position. We call these sequences the \textit{sequences similar to $S$} and they are used to predict the next item of $S$. The process of finding the sequences similar to $S$ is implemented efficiently by using the II. It is done by performing the intersection of the bitsets of the last $x$ items from $S$. The resulting bitset indicates the set of sequences similar to $S$. Using the LT, it is trivial to access these sequences in the PT. For each similar sequence $Y$, the algorithm captures its consequent w.r.t $S$. The \textit{consequent of a sequence $Y$ with respect to a sequence $S$} is the subsequence of $Y$ starting after the last item in common with $S$ until the end of $Y$. Each item of each of those consequents is then stored in a
structure named Count Table (CT). A CT is defined as a hash table with items as keys and a score as corresponding value. This structure holds a list of possible candidate items and their respective score for a specific prediction and hence is unique for each individual prediction task. The item with the highest score within the CT is the predicted item. The primary scoring measure is the support (frequency). But in the case where the support of two items is equal, the confidence is used. We define the support of an item \( s_i \) as the number of times \( s_i \) appears in sequences similar to \( S \), where \( S \) is the sequence to predict. The confidence of an item \( s_i \) is defined as the support of \( s_i \) divided by the total number of training sequences that contain \( s_i \) (the cardinality of the bitset of \( s_i \) in the II). We picked the support as our main scoring measure because it outperformed other measures in terms of accuracy in our experiments.

Performing a prediction is fairly fast. The time complexity is calculated as follows. The search for similar sequences is performed by bitset intersections (the bitwise AND operation), which is \( O(1) \). The construction of the CT is \( O(n) \) where \( n \) is the number of items in all consequents. Finally, choosing the best scoring item is done in \( O(m) \) where \( m \) is the number of unique items in all consequents. In terms of spatial complexity, the CT is the only constructed structure in the prediction process and its hashtable only has \( m \) keys.

**Algorithm 1:** The prediction process of CPT

<table>
<thead>
<tr>
<th>input</th>
<th>( S ): a sequence to predict, ( CPT ): CPT’s structures</th>
</tr>
</thead>
<tbody>
<tr>
<td>output</td>
<td>( x ): the predicted item(s)</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{CT} &= \text{new CountTable();} \\
\text{B} &= \text{findAllBranchesContaining}(S); \\
\text{forall the} \ branch \in B \ \text{do} \\
\quad &\quad \text{c} = \text{getConsequent(branch);} \\
\quad &\quad \text{CT.updateWithItems(c);} \\
\text{end} \\
\text{return CT.getHigestScoringItem();}
\end{align*}
\]
3.3 Optimizations

**Sequence Splitter.** The first optimization is done during the training phase while the PT is being constructed. Let \( \text{splitLength} \) be the maximum allowed length for a sequence. For each sequence longer than \( \text{splitLength} \) items, only the subsequence formed by its last \( \text{splitLength} \) items are inserted in the PT. By using this optimization, the resulting CPT is no longer lossless since sequence information is discarded. Splitting long sequences has for goal to reduce the PT size by reducing the number of possible branches and by enforcing an upper bound on the depth of branches. Intuitively, it may seems that this optimization would negatively affect the prediction’s accuracy. But we have observed that it boosts the accuracy by forcing prediction to focus on the latest \( W \) items of each training sequence. We also observed that this optimization greatly reduces the prediction time and the CPT size (cf. section 4.3).

**Recursive Divider.** One of the problem we experienced early in our research is the low coverage of our approach for prediction. Since our model is based on finding similar sequences that share a fixed subset of items \( T \), if some noise is introduced in \( T \), CPT is only able to find similar sequences containing the same noise. To make our approach more flexible, we introduce a recursive method named the Recursive Divider that tries removing the noise from \( T \) when searching for similar sequences. This approach works by levels \( k = 1, 2... \text{maxLevel} \), where \( \text{maxLevel} \) is a constant indicating the maximum number of levels to explore. At level \( k \), for each subset \( Q \subset T \) such that \( |Q| = k \), the Recursive Divider uses the similar sequences to \( T/Q \) to update the CT. Note that each training sequence is only used once for each level to update the CT. If a prediction cannot be made at level \( k \), the Recursive Divider moves to level \( k + 1 \) if \( k + 1 < \text{maxLevel} \). In the experimentation section, we show that this technique boosts the coverage of CPT.

4 Experimental Evaluation

To evaluate the performance of the proposed prediction model, we performed a set of experiments. Our test environment is made of an Intel i5 third generation processor with 4.5 GB of available RAM on a 64-bit version of Windows8.
Algorithm 2: Recursive divider strategy

input : \( S \): a sequence to predict, \( CPT \): CPT’s structures, \( maxLevel \): order of the RecursiveDivider

output: \( x \): the predicted item(s)

CT = new CountTable();
level = 0;
predicted = null;
while \( predicted == null AND level < maxLevel \) do
    length = \( S \).length - level;
    Q = getSubsequencesOfLength(length);
    for all the subsequence \( \in Q \) do
        B = findAllBranchesContaining(\( S \));
        for all the branch \( \in B \) do
            Item[] c = getConsequent(branch);
            CT.updateWithItems(c);
        end
    end
    predicted = CT.getHighestScoringItem();
end
return predicted;

4.1 Datasets

We used five real-life datasets representing various types of data. Table I-1 summarizes their characteristics. For each dataset, sequences containing less than 3 items were discarded.

BMS is a popular dataset in the field of association rule mining made available for KDD CUP 2000 [16]. It contains web sessions from an e-commerce website, encoded as sequences of integers, representing web pages.
**FIFA** contains web sessions recorded on the 1998 FIFA World Cup Web site and holds over one million web page requests [15]. Originally, the dataset is a set of individual requests containing metadata (e.g. client id and time). We converted requests into sequences by grouping requests by users and splitting a sequence if there was a delay of more than an hour between two requests. Our final dataset is a random sample from the original dataset.

**SIGN** is a dense dataset with long sequences, containing 730 sequences of sign-language utterances transcripted from videos [23].

**KOSARAK** is a dataset containing web sessions from a Hungarian news portal available at [http://fimi.ua.ac.be/data](http://fimi.ua.ac.be/data). It is the largest dataset used in our experimental evaluation.

**BIBLE** is the religious Christian set of books used in plain text as a flow of sentences. The prediction task consists in predicting the next character in a given sequence of characters. The book is split in sentences where each sentence is a sequence. This dataset is interesting since it has a small alphabet with only 75 distinct characters and it is based on natural language.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Sequence count</th>
<th>Unique items</th>
<th>Avg sequence length</th>
<th>Avg item occurrence count per sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMS</td>
<td>15,806</td>
<td>495</td>
<td>6.01</td>
<td>1.00</td>
</tr>
<tr>
<td>FIFA</td>
<td>28,978</td>
<td>3,301</td>
<td>32.11</td>
<td>1.04</td>
</tr>
<tr>
<td>SIGN</td>
<td>730</td>
<td>267</td>
<td>93.00</td>
<td>1.79</td>
</tr>
<tr>
<td>KOSARAK</td>
<td>638,811</td>
<td>39,998</td>
<td>11.64</td>
<td>1.00</td>
</tr>
<tr>
<td>BIBLE</td>
<td>32,529</td>
<td>76</td>
<td>130.96</td>
<td>4.78</td>
</tr>
</tbody>
</table>

### 4.2 Evaluation Framework

We designed a framework in Java to compare our approach with state-of-the-art approaches on all these datasets. The framework is publicly available at [http://goo.gl/GVrjLf](http://goo.gl/GVrjLf) and is developed in Java 7. The following paragraphs describes the evaluation process of our framework.
Each dataset is read in memory. Sequences containing less than three items are discarded. The dataset is then split into a training set and a testing set, using the 10-fold cross-validation technique. For each fold, the training set is used to train each predictor. Once the predictors have been trained, each sequence of the testing set is split into three parts: the context, the prefix and the suffix as shown in Fig. I-3. The prefix and suffix size are determined by two parameters named $PrefixSize (p)$ and $SuffixSize (s)$. The context ($c$) is the remaining part of the sequence and is discarded. For each test sequence, each predictor accepts the prefix as input and makes a prediction. A prediction has three possible outcomes. The prediction is a success if the generated candidate appears in the suffix of the test sequence. The prediction is a no match if the predictor is unable to perform a prediction. Otherwise it is a failure. We define three measures to assess a predictor overall performance. Local Accuracy (eq. 1) is the ratio of successful predictions against the number of failed predictions.

$$Local\_Accuracy = \frac{|successes|}{|successes| + |failures|} \quad (I-1)$$

Coverage (eq. 2) is the ratio of sequence without prediction against the total number of test sequences.

$$Coverage = \frac{|no\_matches|}{|sequences|} \quad (I-2)$$

Accuracy (eq. 3) is our main measure to evaluate the accuracy of a given predictor. It is the number of successful prediction against the total number of test sequences.

$$Accuracy = \frac{|successes|}{|sequences|} \quad (I-3)$$

The above measures are used in our experiments as well as the spatial size (in nodes), the training time (in seconds) and the testing time (in seconds). The spatial size is calculated in nodes because the spatial complexity of all predictors can be represented in terms of nodes. This measure is meant to show the spatial complexity and is not used to determine the exact size of a model.
Table I-2 Experiment’s parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequenceMinSize</td>
<td>(\mathbb{Z})</td>
<td>Minimum size for a sequence</td>
</tr>
<tr>
<td>sequenceMaxSize</td>
<td>(\mathbb{Z})</td>
<td>Maximum size for a sequence</td>
</tr>
<tr>
<td>consequentSize</td>
<td>(\mathbb{Z})</td>
<td>Suffix size for the prediction</td>
</tr>
<tr>
<td>windowSize</td>
<td>(\mathbb{Z})</td>
<td>Prefix size for the prediction</td>
</tr>
</tbody>
</table>

Table I-3 Parameters of CPT

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>splitLength</td>
<td>(\mathbb{Z})</td>
<td>max tree height</td>
</tr>
<tr>
<td>maxLevel</td>
<td>(\mathbb{Z})</td>
<td>Order of the RecursiveDivider</td>
</tr>
</tbody>
</table>

4.3 Experiments

Overall performance. The goal of the first experiment consists in getting an overview of the performances (accuracy, training and testing time and space) of CPT against DG, 1st order PPM and All-\(K\)th-Order Markov (AKOM). DG and AKOM were respectively tuned with a lookahead window of 4 and with an order of 5, since these values gave the best performance and are typically good values for these algorithms \([2, 3, 18]\). Results are shown in Tables I-4 and I-5. They show that CPT yields a higher accuracy for all but one dataset. DG and PPM perform well in some situations but CPT is more consistent across all datasets. The training time, just like the testing time can be critical for some applications. In this experiment, CPT is always faster to train than DG and All-\(K\)th-Order Markov by at least a factor of 3, and has comparable training time to PPM. The downside of CPT is that making a prediction can take longer than other methods. This characteristic is a trade off for a higher accuracy and is mainly caused by the Recursive Divider optimization described in section 3.3. The coverage is not presented because a high coverage (> 95%) is achieved by all the predictor for all datasets and it is also indirectly included in the overall accuracy measure.

Scalability. Our second experiment compares the scalability of each approach. The importance
### Table I-4 Comparison of accuracy and model size

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Overall Accuracy</th>
<th>Size (nodes)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DG</td>
<td>CPT</td>
</tr>
<tr>
<td>BMS</td>
<td>36.07</td>
<td><strong>38.45</strong></td>
</tr>
<tr>
<td>FIFA</td>
<td>25.87</td>
<td><strong>37.2</strong></td>
</tr>
<tr>
<td>SIGN</td>
<td>3.54</td>
<td><strong>34.795</strong></td>
</tr>
<tr>
<td>KOSARAK</td>
<td>31.44</td>
<td><strong>34.26</strong></td>
</tr>
<tr>
<td>BIBLE</td>
<td>6.26</td>
<td>82.06</td>
</tr>
</tbody>
</table>

### Table I-5 Comparison of training time and testing time

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Training time (s)</th>
<th>Testing time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DG</td>
<td>CPT</td>
</tr>
<tr>
<td>BMS</td>
<td>0.076</td>
<td>0.018</td>
</tr>
<tr>
<td>FIFA</td>
<td>3.032</td>
<td>0.153</td>
</tr>
<tr>
<td>SIGN</td>
<td>0.172</td>
<td><strong>0.008</strong></td>
</tr>
<tr>
<td>KOSARAK</td>
<td>9.697</td>
<td>0.741</td>
</tr>
<tr>
<td>BIBLE</td>
<td>0.803</td>
<td><strong>0.007</strong></td>
</tr>
</tbody>
</table>
of scalability is application specific. But it is an important factor for most prediction tasks since the
ability to scale a prediction model can directly or indirectly limit its accuracy and coverage. For this
experiment, we used the FIFA dataset because of its high number of sequences and unique items.
The experiment is conducted in steps, where the predictors are trained and tested with a higher
number of sequences at each following step. Figure I-4 shows the results in terms of accuracy,
space and time. All training times in this experiment follow a linear evolution, but CPT and PPM
operate at a much lower scale and are very close. PPM and DG have low spatial complexity because
of their compact representation compared to CPT which takes more place but still grows linearly.

**Figure I-4** Comparison of scalability

**Prefix length.** The third experiment assesses the effect of the prefix size on the accuracy and
coverage. Results are shown in Figure I-5 for the FIFA dataset. Recall that the predictors output a
prediction based on the prefix given as input. The longer the prefix, the more contextual information
is given to the predictor. Note that DG, PPM and All-Kth-Order Markov use a predetermined
portion of the prefix defined by the order of each algorithm. Thus, by increasing the prefix length,
we can observe that none of these algorithms get an increase in any performance measures. CPT
takes advantage of a longer prefix by finding more precise (longer) patterns in its prediction tree,
to yield a higher accuracy. The accuracy of CPT gets higher as the prefix length is raised. But after
the prefix reaches a length of 8 (specific to the dataset), the accuracy decreases. This is because the
algorithm may not be able to match a given prefix to any branches in the prediction tree. It means
that this parameter should be finely tuned for each dataset to maximize the accuracy. The figure on
the right of Fig. I-5 shows the influence of the prefix length on the CPT spatial complexity.

**Figure I-5** Influence of prefix length on accuracy and model size

**Optimizations.** The fourth experiment assesses the influence of the Recursive Divider optimi-
ization (cf. Section 3.3) for CPT. The Recursive Divider aims at boosting the coverage of predic-
tions using the CPT by ignoring items that could be noise during the prediction process. But it also
indirectly influences accuracy. Figure I-6 shows the effect of the Recursive Divider on the FIFA
dataset by sequentially incrementing the `maxLevel` parameter. We can observe that the accuracy
and the coverage of CPT are getting higher as the `maxLevel` parameter is raised. Also, the coverage
and the accuracy measures quickly stabilize without affecting the testing time. This strategy’s pa-
rameter can therefore be set to a really high value to guarantee the best coverage and accuracy and
it does not need to be adjusted for each dataset.

The fifth experiment measures the influence of the Sequence Splitter optimization (cf. Section
3.3). It truncates long sequences before they are inserted in the prediction tree during the training
phase. This makes the prediction tree more compact by reducing the number of possible branches
and their depth. Reducing the depth improves the time complexity for both the training and testing
processes. In this experiment we used the FIFA dataset because it has long sequences. We evaluated
the performance of our model against different values for the `splitLength` parameter. For low values
(eg. 5) most of the training sequences are split. By setting `splitLength` to a high value (eg. 40 or
Figure I-6 Influence of the Recursive Divider optimization

more), only a small number of sequences are splitted. In Figure I-7, we show the effect of applying the Sequence Splitter strategy on the accuracy, the spatial size and the testing time, for various split lengths. By setting \textit{splitLength} to a low value (left side of each chart of Fig. I-7), the spatial size is reduced by a factor of 7 while having a really low training and testing time and still having a high accuracy. Once again, this parameter should be finely tuned for each dataset in order to achieve the best performance.

Figure I-7 Influence of the Sequence Splitter optimization
5 Conclusion

Predicting the next item of a sequence over a finite alphabet is essential to a wide range of applications in many domains. In this paper we present a novel prediction model named the Compact Prediction Tree for sequence prediction. CPT is lossless (it can use all the information from training sequences to make a prediction), its construction process has a low time complexity. We also present two optimizations (Recursive Divider and Sequence Splitter), which respectively boost the coverage of CPT and reduce its size.

We compared CPT to state-of-the-art approaches, namely PPM, All-$K$th-Order Markov Model and DG on six real-life datasets. The source code of algorithms and datasets used in the experiments are available at http://goo.gl/GVrjLf. Results show that CPT achieves the highest accuracy on all but one dataset with an accuracy up to $12\%$ higher than the second best approach. CPT also shows better training time than DG and All-$K$th Order Markov Model by at least a factor of 3. CPT is also considerably smaller than the All-$K$th Order Markov model by at least a factor of 2. CPT is easily adaptable for different applications and contexts as shown in the experiments.

In the future, we aim to further improve the accuracy of CPT and its compression. We believe that higher compression can be achieved by grouping patterns of nodes and pruning nodes in the prediction tree. We also plan to compare our model against other prediction techniques such as Context Tree Weighting and Neural Networks.
CHAPITRE II

CPT+: DECREASING THE TIME/SPACE COMPLEXITY OF THE COMPACT PREDICTION TREE

RÉDUCTION DE LA COMPLEXITÉ SPATIALE ET TEMPORELLE DU COMPACT PREDICTION TREE POUR LA PRÉDICTION DE SÉQUENCES
Abstract

Predicting next items of sequences of symbols has many applications in a wide range of domains. Several sequence prediction models have been proposed such as DG, All-k-order markov and PPM. Recently, a model named Compact Prediction Tree (CPT) has been proposed. It relies on a tree structure and a more complex prediction algorithm to offer considerably more accurate predictions than many state-of-the-art prediction models. However, an important limitation of CPT is its high time and space complexity. In this article, we address this issue by proposing three novel strategies to reduce CPT’s size and prediction time, and increase its accuracy. Experimental results on seven real life datasets show that the resulting model (CPT+) is up to 98 times more compact and 4.5 times faster than CPT, and has the best overall accuracy when compared to six state-of-the-art models from the literature: All-K-order Markov, CPT, DG, Lz78, PPM and TDAG.

Keywords: sequence prediction, next item prediction, accuracy, compression.

1 Introduction

Sequence prediction is an important task with many applications [14, 26]. Let be an alphabet of items (symbols) $Z = \{e_1, e_2, ..., e_m\}$. A sequence $s$ is an ordered list of items $s = \langle i_1, i_2, ..., i_n \rangle$, where $i_k \in Z \ (1 \leq k \leq n)$. A prediction model is trained with a set of training sequences. Once trained, the model is used to perform sequence predictions. A prediction consists in predicting the next items of a sequence. This task has numerous applications such as web page prefetching [2,18], consumer product recommendation [9], weather forecasting [10], and stock market prediction [11].

Many sequence predictions models have been proposed. One of the most popular is Prediction by Partial Matching (PPM) [1]. It is based on the Markov property and has inspired a multitude of other models such as Dependency Graph (DG) [2], All-K-Order-Markov (AKOM) [3], Transition Directed Acyclic Graph (TDAG) [4], Probabilistic Suffix Tree (PST) [14] and Context Tree Weighting (CTW) [14]. Although much work has been done to reduce the temporal and spatial complexity of these models (e.g. [14, 18]), few work attempted to increase their accuracy. Besides, several compression algorithms have been adapted for sequence predictions such as LZ78 [12] and
Active Lezi [7]. Moreover, machine learning algorithms such as neural networks and sequential rule mining have been applied to perform sequence prediction [20, 26, 27].

However, these models suffer from some important limitations [21]. First, most of them assume the Markovian hypothesis that each event only depends on the previous events. If this hypothesis does not hold, prediction accuracy using these models can severely decrease [18, 21]. Second, all these models are built using only part of the information contained in training sequences. Thus, these models do not use all the information contained in training sequences to perform predictions, and this can severely reduce their accuracy. For instance, Markov models typically consider only the last \( k \) items of training sequences to perform a prediction, where \( k \) is the order of the model. One may think that a solution to this problem is to increase the order of Markov models. However, increasing the order of Markov models often induces a very high state complexity, thus making them impractical for many real-life applications [18].

CPT [21] is a recently proposed prediction model which compresses training sequences without information loss by exploiting similarities between subsequences. It has been reported as more accurate than state-of-the-art models PPM, DG, AKOM on various real datasets. However, a drawback of CPT is that it has an important spatial complexity and a higher prediction time than these models. Therefore, an important research problem is to propose strategies to reduce the size and prediction time of CPT. Reducing the spatial complexity is a very challenging task. An effective compression strategy should provide a huge spatial gain while providing a minimum overhead in terms of training time and prediction time. Furthermore, it should also preserve CPT’s lossless property to avoid a decrease in accuracy. Reducing prediction time complexity is also very challenging. An effective strategy to reduce prediction time should access as little information as possible for making predictions to increase speed, but at the same time it should carefully select this information to avoid decreasing accuracy.

In this paper, we address these challenges by proposing three strategies named FSC (Frequent Subsequence Compression), SBC (Simple Branches Compression) and PNR (Prediction with improved Noise Reduction). The two first strategies are compression strategies that reduce CPT size by up to two orders of magnitude while not affecting accuracy. The third strategy reduces the pre-
prediction time by up to 4.5 times and increases accuracy by up to 5%. This paper is organized as follows. Section 2 introduces CPT. Sections 3 and 4 respectively describes the two compression strategies (FSC and SBC), and the prediction time reduction strategy (FNR). Section 5 presents an experimental evaluation of each strategy on seven real datasets against five state-of-the-art prediction models. Finally, Section 6 draws conclusion.

2 Compact Prediction Tree

The Compact Prediction Tree (CPT) is a recently proposed prediction model [21]. Its main distinctive characteristics w.r.t other prediction models are that (1) CPT stores a compressed representation of training sequences with no loss or a small loss, and (2) CPT measures the similarity of a sequence to the training sequences to perform a prediction. The similarity measure is noise tolerant and thus allows CPT to predict the next items of subsequences that have not been previously seen in training sequences, whereas other proposed models such as PPM and All-K-order-markov cannot perform prediction in such case.

The training process of CPT takes as input a set of training sequences and generates three distinct structures: (1) a Prediction Tree (PT), (2) a Lookup Table (LT) and (3) an Inverted Index. During training, sequences are considered one by one to incrementally build these three structures. For instance, Fig. II-1 illustrates the creation of the three structures by the successive insertions of sequences $s_1 = \langle A, B, C \rangle$, $s_2 = \langle A, B \rangle$, $s_3 = \langle A, B, D, C \rangle$, $s_4 = \langle B, C \rangle$ and $s_5 = \langle E, A, B, A \rangle$, where the alphabet $Z = \{A, B, C, D, E\}$ is used. The Prediction Tree is a type of prefix tree (aka trie). It contains all training sequences. Each tree node represents an item and each training sequence is represented by a path starting from the tree root and ending by an inner node or a leaf. Just like a prefix tree, the prediction tree is a compact representation of the training sequences. Sequences sharing a common prefix share a common path in the tree. The Lookup Table is an associative array which allows to locate any training sequences in the prediction tree with a constant access time. Finally the Inverted Index is a set of bit vectors that indicates for each item $i$ from the alphabet $Z$, the set of sequences containing $i$.

CPT’s prediction process relies on the three aforementioned data structures. For a sequence
Figure II-1 Building CPT structures

$s = \langle i_1, i_2, ... i_n \rangle$ of $n$ elements, the suffix of $s$ of size $y$ with $1 \leq y \leq n$ is defined as $P_y(s) = \langle i_{n-y+1}, i_{n-y+2} ... i_n \rangle$. Predicting the next items of $s$ is performed by identifying the sequences similar to $P_y(s)$, that is the sequences containing all items in $P_y(s)$ in any order. The suffix length is a parameter similar to the model’s order of All-k-order Markov and DG. Identifying the optimal value is done empirically by starting with a length of 1. CPT uses the consequent of each sequence similar to $s$ to perform the prediction. Let $u = \langle j_1, j_2, ... j_m \rangle$ be a sequence similar to $s$. The consequent of $u$ w.r.t to $s$ is the longest subsequence $\langle j_v, j_{v+1}, ... j_m \rangle$ of $u$ such that $\bigcup_{k=1}^{v-1}\{j_k\} \subseteq P_y(s)$ and $1 \leq v \leq m$. Each item found in the consequent of a similar sequence of $s$ is stored in a data structure called Count Table (CT). The count table stores the support (frequency) of each of these.
items, which is an estimation of $P(e|P_y(s))$. CPT returns the most supported item(s) in the CT as its prediction(s).

The similarity measure in CPT is initially strict for each prediction task but is dynamically loosened to ignore noise. Identifying similar sequences, and more particularly the noise avoidance strategy of CPT, is very time consuming and accounts for most of CPT’s prediction time [21]. For a given sequence, if CPT cannot find enough similar sequences to generate a prediction, it will implicitly assume the sequence contains some noise. The prediction process is then repeated but with one or more items omitted from the given sequence. CPT’s definition of noise is implicit and has for sole purpose to ensure that a prediction can be made every time.

3 Compression strategies

CPT has been presented as one of the most accurate sequence prediction model [21] but its high spatial complexity makes CPT unsuitable for applications where the number of sequences is very large. CPT’s size is smaller than All-k-Order Markov and TDAG but a few orders of magnitude larger than popular models such as DG and PPM. CPT’s prediction tree is the largest data structure and account for most of its spatial complexity. In this section, we focus on strategies to reduce the prediction tree’s size.

3.1 Frequent subsequence compression (FSC)

In a set of training sequences, frequently occurring subsequences of items can be found. For some datasets, these subsequences can be highly frequent. The FSC strategy identifies these frequent subsequences and replaces each of them with a single item. For a sequence $s = \langle i_1, i_2, ..., i_n \rangle$. A sequence $c = \langle i_{m+1}, i_{m+2}, ..., i_{m+k} \rangle$ is a subsequence of $s$, denoted as $c \sqsubseteq s$, iff $1 \leq m \leq m + k \leq n$. For a set of training sequences $S$, a subsequence $d$ is considered a frequent subsequence iff $|\{t | t \in S \land d \sqsubseteq t\}| \geq \text{minsup}$ for a minimum support threshold $\text{minsup}$ defined per dataset.

Frequent subsequences compression is done during the training phase and is performed in three
steps: (1) identification of frequent subsequences in the training sequences, (2) generation of a new item in the alphabet $Z$ of items for each frequent subsequence, and (3) replacement of each frequent subsequence by the corresponding new item when inserting training sequences in the prediction tree. Identifying frequent subsequences in a set of sequences is a known problem in data mining for which numerous approaches have been proposed. In FSC, we use the well known PrefixSpan [24] algorithm. PrefixSpan is one of the most efficient sequential pattern mining algorithm. It has been adapted by incorporating additional constraints to fit the problem of sequence prediction. Subsequences have to be contiguous, larger than a minimum length $minSize$ and shorter than a maximum length $maxSize$. Both $minSize$ and $maxSize$ are parameters of this compression strategy that are defined per application.

A new data structure, Subsequence Dictionary (DCF), is introduced to store the frequent subsequences. This dictionary associates each frequent subsequence with its corresponding item. The DCF offers a fast way to translate each subsequence into its respective item and vice-versa, in $O(1)$ time. When inserting training sequences into the prediction tree, the DCF is used to replace known frequent subsequences with single items. For example, figure II-2 illustrates the resulting prediction tree after applying FSC to the tree shown in Fig. II-1. The frequent subsequence $\langle A, B \rangle$ has been replaced by a new symbol $x$, thus reducing the number of nodes in the prediction tree. The FSC compression strategy influences the shape of the prediction tree by reducing its height and number of nodes. With respect to the prediction process, FSC only influences execution time. The additional cost is the on-the-fly decompression of the prediction tree, which is fast and non intrusive because of the DCF structure.

3.2 Simple Branches Compression (SBC)

Simple Branches Compression is an intuitive compression strategy that reduces the size of the prediction tree. A simple branch is a branch leading to a single leaf. Thus, each node of a simple branch has between 0 and 1 children. The SBC strategy consists of replacing each simple branch by a single node representing the whole branch. For instance, part (2) of Fig. II-2 illustrates the prediction tree obtained by applying the DCF and SBC strategies for the running example. The SBC strategy has respectively replaced the simple branches $D, C, B, C$ and $E, x, A$ by single nodes.
Figure II-2 Application of the FSC and SBC compression strategies

DC, BC and ExA. Identifying and replacing simple branches is done by traversing the prediction tree from the leaves using the inverted index. Only the nodes with a single child are visited. Since the Inverted Index and Lookup Table are not affected by this strategy, the only change that needs to be done to the prediction process is to dynamically uncompress nodes representing simple branches when needed.

4 Time reduction strategy

Strategy 3: Prediction with improved Noise Reduction (PNR). As previously explained, to predict the next item $i_{n+1}$ of a sequence $s = \langle i_1, i_2, ..., i_n \rangle$, CPT uses the suffix of size $y$ of $s$ denoted as $P_y(s)$ (the last $y$ items of $s$), where $y$ is a parameter that needs to be set for each dataset. CPT predicts the next item of $s$ by traversing the sequences that are similar to its suffix $P_y(s)$. Searching for similar sequences is very fast ($O(y)$). However, the noise reduction mechanism used for prediction (described in Section 2) is not. The reason is that it considers not only $P_y(s)$ to perform a prediction, but also all subsequences of $P_y(s)$ having a size $t > k$, where $k$ is a parameter. The larger $y$ and $k$ are, the more subsequences need to be considered, and thus the more the prediction time increases.

For a prediction task, items in a training sequence may be considered as noise if their sole pres-
ence negatively impacts a prediction’s outcome. The PNR strategy is based on the hypothesis that noise in training sequences consists of items having a low frequency, where an item’s frequency is defined as the number of training sequences containing the item. For this reason, PNR removes only items having a low frequency during the prediction process. Because the definition of noise used in CPT+ is more restrictive than that of CPT, a smaller number of subsequences are considered. This reduction has a positive and measurable impact on the execution time, as it will be demonstrated in the experimental evaluation (see Section 5).

The PNR strategy (Algorithm 3) takes as parameter the prefix $P_y(s)$ of a sequence to be predicted $s$, CPT’s structures and the noise ratio $TB$ and a minimum number of updates, $MBR$, to be performed on the count table (CT) to perform a prediction. The noise ratio $TB$ is defined as the percentage of items in a sequence that should be considered as noise. For example, a noise ratio of 0 indicates that sequences do not contain noise, while a ratio of 0.2 means that 20% of items in a sequence are considered as noise. PNR is a recursive procedure. To perform a prediction, we require that PNR considers a minimum number of subsequences derived from $P_y(s)$. PNR first removes noise from each subsequence. Then, the CT is updated using these subsequences. When the minimum number of updates is reached, a prediction is performed as in CPT using the CT.

The PNR strategy is a generalization of the noise reduction strategy used by CPT. Depending on how the parameters are set, PNR can reproduce the behavior of CPT’s noise reduction strategy. The three main contributions brought by PNR are to require a minimum number of updates on the CT to perform a prediction, to define noise based on the frequency of items, and to define noise proportionally to a sequence length. Finding the appropriate values for both $TB$ and $MBR$ can be achieved empirically.

5 Experimental evaluation

We have performed several experiments to compare the performance of CPT+ against five state-of-the-art sequence prediction models: All-K-order Markov, DG, Lz78, PPM and TDAG. We picked these models to match the models used in the original paper describing CPT [21] and added both Lz78 and TDAG. To implement CPT+, we obtained and modified the original source code of CPT [21]. To allow reproducing the experiments, the source code of the prediction models and
Algorithm 3: The prediction algorithm using PNR

input : $P_y(s)$: a sequence suffix, $CPT$: CPT’s structures, $TB$: a noise ratio, $MBR$: minimum number of CT updates

output: $x$: the predicted item(s)

queue.add($P_y(s)$);

while $updateCount < MBR \land queue.notEmpty()$ do

suffix = queue.next();

noisyItems = selectLeastFrequentItems($TB$);

foreach noisyItem $\in$ noisyItems do

suffixWithoutNoise = removeItemFromSuffix(suffix, noisyItem);

if $suffixWithoutNoise.length > 1$ then

queue.add(suffixWithoutNoise);

end

updateCountTable($CPT.CT$, suffixWithoutNoise);

updateCount++;

end

return performPrediction($CPT.CT$);

end

datasets are provided at http://goo.gl/GVrjLf. All models are implemented using Java 8.
Experiments have been performed on a computer with a dual core 4th generation Intel i5 with 8 GB RAM and a SSD drive connected with SATA 600. For all prediction models, we have empirically attempted to set their parameters to optimal values. PPM and LZ78 do not have parameters. DG and AKOM have respectively a window of four and an order of five. To avoid consuming an excessive amount of memory, TDAG has a maximum depth of 7. CPT has two parameters, $splitLength$ and $maxLevel$ and CPT+ has six parameters; three for the FSC strategy, two for the PNR strategy and $splitLength$ from CPT. The values of these parameters have also been empirically found and are provided in the project source code. Experiment specific parameters are the minimum and maximum length of sequences used, the number of items to be considered to perform a prediction (the suffix length), and the number of items used to verify the prediction (called the consequent length).
Let be a sequence \( s = (i_1, i_2, \ldots, i_n) \) having a suffix \( S(s) \) and a consequent \( C(s) \). Each model takes the suffix as input and outputs a predicted item \( p \). A prediction is deemed successful if \( p \) is the first item of \( C(s) \). Datasets having various characteristics have been used (see Table II-1) such as short/long sequences, sparse/dense sequences, small/large alphabets and various types of data. The BMS, Kosarak, MSNB and FIFA datasets consist of sequences of webpages visited by users on a website. In this scenario, prediction models are applied to predict the next webpage that a user will visit. The SIGN dataset is a set of sentences in sign language transcribed from videos. Bible Word and Bible Char are two datasets originating from the Bible. The former is the set of sentences divided into words. The latter is the set of sentences divided into characters. In both datasets, a sentence represents a sequence.

To evaluate prediction models, a prediction can be either a success if the prediction is accurate, a failure if the prediction is inaccurate or a no match if the prediction model is unable to perform a prediction. Four performance measures are used in experiments: Coverage is the ratio of sequences without prediction against the total number of test sequences. Accuracy is the number of successful predictions against the total number of test sequences. Training time is the time taken to build a model using the training sequences Testing time is the time taken to make a prediction for all test sequences using a model.

<table>
<thead>
<tr>
<th>Name</th>
<th>Sequence count</th>
<th>Distinct item count</th>
<th>Average length</th>
<th>Type of data</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMS</td>
<td>15,806</td>
<td>495</td>
<td>6.01</td>
<td>webpages</td>
</tr>
<tr>
<td>KOSARAK</td>
<td>638,811</td>
<td>39,998</td>
<td>11.64</td>
<td>webpages</td>
</tr>
<tr>
<td>FIFA</td>
<td>573,060</td>
<td>13,749</td>
<td>45.32</td>
<td>webpages</td>
</tr>
<tr>
<td>MSNB</td>
<td>250,697</td>
<td>17</td>
<td>3.28</td>
<td>webpages</td>
</tr>
<tr>
<td>SIGN</td>
<td>730</td>
<td>267</td>
<td>93.00</td>
<td>language</td>
</tr>
<tr>
<td>BIBLE Word</td>
<td>42,436</td>
<td>76</td>
<td>18.93</td>
<td>sentences</td>
</tr>
<tr>
<td>BIBLE Char</td>
<td>32,502</td>
<td>75</td>
<td>128.35</td>
<td>characters</td>
</tr>
</tbody>
</table>
**Experiment 1: Optimizations comparison.** We have first evaluated strategies that aim to reduce the space complexity of CPT (cf. Section 3) by measuring the compression rate and the amount of time spent for training. Other measures such as prediction time, coverage and accuracy are not influenced by the compression. For a prediction tree $A$ having $s$ nodes before compression and $s_2$ nodes after compression, the compression rate $tc_A$ of $A$ is defined as $tc = 1 - (s_2/s)$, a real number in the $[0,1]$ interval. A larger value means a higher compression. The two strategies are first evaluated separately (denoted as FSC and SBC) and then together (denoted as CPT+). Compression provides a spatial gain but increases execution time. Figure II-3 illustrates this relationship for each compression strategy.

![Compression rate and training time of the compression strategies](image)

It can also be observed in Fig. II-3 (left) that the compression rate varies depending on the dataset from 58.90% to 98.65%. FSC provides an average compression rate of 48.55% with a small standard deviation (6.7%) and SBC offers an average compression rate of 77.87% with a much larger standard deviation (15.9%). For each tested dataset, SBS has a compression rate very similar to CPT+ with the exception of MSNBC and Bible Char. It accounts for most of CPT+ compression rate and thus making FSC relevant only in applications requiring a smaller model. MSNBC is the least affected by the compression strategies. The reason is that MSNBC has a very small alphabet and thus that the tree is naturally compressed because its branches are highly overlapping. In fact, MSNBC has only 17 distinct items, although the lengths of its sequences are similar to those of the other datasets. The dataset where the SBC and SFC strategies provide the highest compression rate is SIGN. Even though SIGN contains a small number of sequences, each sequence is very long (an average of 93 items). It causes the branches of SIGN’s prediction tree to rarely overlap, and a large
amount of its nodes only have a single child. SIGN is a very good candidate for applying the SBC strategy. Using only SBC, a compression rate of 98.60% is attained for SIGN.

Figure II-3 also illustrates the training time for the FSC and SBC strategies. It is measured as a multiplicative factor of CPT’s training time. For example, a factor $x$ for SBC means that the training time using SBC is $x$ times longer than that of CPT without SBC. A small factor, close to one, means the additional training time is small. It is interesting to observe how the training time when both strategies are combined is less than the sum of their respective training time. Although SBC and FSC are independently applied to CPT, SBC reduces the execution time of FSC because less branches have to be compressed. Overall, SBC has a small impact on the training time while providing most of the compression gain, this makes SBS the most profitable strategy. While SFC has a higher impact on the training time, SFC enhances the compression rate for each dataset.

We also evaluated the performance improvement in terms of execution time and accuracy obtained by applying the PNR strategy. Fig. II-4 (left) compares the prediction time of CPT+ (with PNR) with that of CPT. The execution time is reduced for most datasets and is up to 4.5 times smaller for SIGN and MSNBC. For Bible Word and FIFA, prediction times have increased but a higher accuracy is obtained, as shown in Fig. II-4 (right). The gain in prediction time for CPT+ is dependent on both PNR and CPT parameters. This gain is thus dataset specific and non linear because of the difference in complexity of CPT and CPT+. The influence of PNR on accuracy is positive for all datasets except MSNBC. For Bible Char, the improvement is as high as 4.38%. PNR is thus a very effective strategy to reduce the prediction time while providing an increase in prediction accuracy.

![Figure II-4](image_url) Execution time and accuracy gains provided by the PNR strategy
Experiment 2: Scalability. In this experiment we compared the spatial complexity of CPT+ (with both compression strategies) against CPT, All-K-order Markov, DG, Lz78, PPM and TDAG. Only the FIFA and Kosarak datasets were used in this experiment because of their large number of sequences. In Fig. II-5, the spatial size of each model is evaluated against a quadratically growing set of training sequences - up to 128,000 sequences. Both PPM and DG have a sub linear growth which makes them suitable for large datasets. CPT+’s growth is only an order of magnitude larger than PPM and DG and a few orders less than CPT, TDAG and LZ78. The compression rate of CPT+ tends to slightly diminish as the number of sequences grows. This is due to more branches overlapping in the prediction tree, a phenomenon that can generally be observed in tries.

An interesting observation is that for both datasets, when the number of training sequences is smaller than the number of items, CPT+ has a smaller footprint than the other prediction models. For the FIFA dataset, when 1000 sequences are used, CPT+’s node count is 901 compared to the 1,847 unique items in the alphabet. Results are similar for Kosarak. Models such as PPM and DG can’t achieve such a small footprint in these use cases because they have a least one node per unique item.

![Figure II-5 Scalability of the prediction models.](image)

Experiment 3: Comparison with other prediction models. In experiment 1, we have compared the prediction accuracy of CPT+ and CPT to assess the improvement obtained by applying the PNR strategy. In this experiment, we compare the accuracy of CPT+ with five state-of-the-art prediction models commonly used in the literature, All-K-order Markov, DG, Lz78, PPM et TDAG, on the same datasets. Each prediction model has been trained and tested using $k$-fold cross validation with $k = 14$ to obtain a low variance for each run. Table II-2 shows the prediction accuracy
obtained by each model. Results indicates that CPT+ offers a generally higher accuracy than the compared models from the literature while also being more consistent across the various datasets.

Table II-2 Prediction models and their accuracy

<table>
<thead>
<tr>
<th>Datasets</th>
<th>CPT+</th>
<th>CPT</th>
<th>AKOM</th>
<th>DG</th>
<th>LZ78</th>
<th>PPM</th>
<th>TDAG</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMS</td>
<td>38.25</td>
<td>37.90</td>
<td>31.26</td>
<td>36.46</td>
<td>33.46</td>
<td>31.06</td>
<td>6.95</td>
</tr>
<tr>
<td>SIGN</td>
<td>33.01</td>
<td>32.33</td>
<td>8.63</td>
<td>3.01</td>
<td>4.79</td>
<td>4.25</td>
<td>0.00</td>
</tr>
<tr>
<td>MSNBC</td>
<td>61.50</td>
<td>61.64</td>
<td>47.88</td>
<td>55.68</td>
<td>43.64</td>
<td>38.06</td>
<td>31.14</td>
</tr>
<tr>
<td>Bible word</td>
<td>27.52</td>
<td>22.05</td>
<td>38.68</td>
<td>24.92</td>
<td>27.39</td>
<td>27.06</td>
<td>23.33</td>
</tr>
<tr>
<td>Bible char</td>
<td>73.52</td>
<td>69.14</td>
<td>7.96</td>
<td>0.00</td>
<td>3.02</td>
<td>0.10</td>
<td>9.90</td>
</tr>
<tr>
<td>Kosarak</td>
<td>37.64</td>
<td>33.82</td>
<td>20.52</td>
<td>30.82</td>
<td>20.50</td>
<td>23.86</td>
<td>1.06</td>
</tr>
<tr>
<td>FIFA</td>
<td>35.94</td>
<td>34.56</td>
<td>25.88</td>
<td>24.78</td>
<td>24.64</td>
<td>22.84</td>
<td>7.14</td>
</tr>
</tbody>
</table>

6 Conclusion

In this paper we present CPT+, a variation of CPT that includes two novel compression strategies (FSC and SBC) to reduce its size and one strategy to improve prediction time and accuracy (PNR). Experimental results on seven real datasets show that CPT+ is up to 98 times smaller than CPT, performs predictions up to 4.5 times faster, and is up to 5% more accurate. A comparison with six state-of-the-art sequence prediction models (CPT, All-K-Order Markov, DG, Lz78, PPM and TDAG) shows that CPT+ is on overall the most accurate model. To allow reproducing the experiments, the source code of the prediction models and datasets are provided at http://goo.gl/GVrjLf.
CONCLUSION GÉNÉRALE

Dans le cadre de ce travail, nous avons introduit Compact Prediction Tree (CPT), un modèle pour la prédiction de séquences sur un alphabet fini. CPT se distingue des précédents modèles de prédictions de séquences par sa compression des séquences d’entraînement et sa tolérance au bruit. En effet, CPT est un algorithme de compression sans perte et qui utilise les séquences d’entraînement entières lors de prédictions. Sa tolérance au bruit provient du fait que CPT utilise une mesure de similarité dynamique pour identifier des séquences importantes lors du processus de prédiction.

Le modèle CPT introduit deux stratégies ; Recursive Divider et Sequence Splitter. Celles-ci permettent d’augmenter l’exactitude de CPT et de diminuer la taille de l’arbre de prédiction respectivement. Nous avons évalué les performances de CPT contre trois modèles de prédiction sur six jeux de données réels. Nous nous sommes intéressés à quatre mesures de performance : l’exactitude, la taille, le temps d’entraînement et le temps de prédiction. CPT obtient une meilleure exactitude que les autres modèles sur cinq des six jeux de données tout en ayant des temps d’entraînement très faibles et comparables à ceux de PPM ; PPM étant une simple implémentation des processus de Markov de premier ordre. Néanmoins, la complexité spatiale de CPT demeure plutôt grande (elle est plus importante que celle de DG et PPM, bien qu’elle soit considérablement plus petite que celle de AKOM et TDAG, selon nos expériences). Une taille spatiale aussi grande peut rendre CPT inutilisable pour certaines applications ayant des restrictions de mémoire. De plus, CPT a une importante complexité temporelle pour la prédiction. Le temps nécessaire pour que CPT fasse un ensemble de prédictions est beaucoup plus élevé que pour tous les autres modèles utilisés, selon notre expérimentation. L’utilisation de la stratégie Recursive Divider impose un important coût temporel initial lors du processus de prédiction. Cette stratégie a cependant un remarquable effet sur la couverture et l’exactitude des prédications de CPT.

Pour pallier ces limites, nous avons introduit une seconde version de CPT nommée CPT+, qui incorpore trois stratégies additionnelles. Les deux premières, CCF et CBS, servent à compresser l’arbre de prédiction sans affecter le processus de prédiction et n’ont donc aucun impact sur l’exactitude et la couverture de CPT. Ces deux stratégies combinées rendent CPT jusqu’à 98 fois
plus petit. CCF a un taux de compression inférieur à CBS et a un coût temporel très élevé. Nous concluons que CBS est une stratégie essentielle de CPT et que l’utilisation de CCF est seulement recommandée pour les applications ayant d’importantes contraintes spatiales. La troisième stratégie intégrée à CPT+ est nommée PNR. Elle est un remplacement et une généralisation de la stratégie *Recursive Divider* présentée de CPT. PNR réduit les temps de prédiction de CPT jusqu’à 4,5 fois tout en augmentant l’exactitude de CPT pour certains jeux de données. PNR est elle aussi une stratégie essentielle à CPT+ ; le gain en temps de prédiction est important.

Nous avons comparé CPT+ à CPT et à cinq autres modèles. Nos résultats montrent que CPT+ offre une plus haute exactitude dans la plupart des jeux de données utilisés tout en ayant des temps d’exécution nettement plus faible que CPT. Ceux-ci sont tout de même plus importants que les modèles PPM et DG. Nous avons aussi comparé la taille de ces modèles et CPT+ est plus petit par plusieurs ordres de magnitude que CPT, AKOM, TDAG et LZ78. Dans nos expériences, CPT+ est aussi plus petit que PPM et DG lorsque le nombre de séquences est inférieur à la cardinalité de l’alphabet du jeu de données. Cette dernière caractéristique est particulièrement intéressante pour l’utilisation de CPT+ comme un modèle de compression.

Les processus d’entraînement et de prédiction de CPT sont constitués d’ensembles d’opérations indépendantes qui pourraient être effectuées en parallèle. Il serait donc intéressant d’utiliser CPT dans un environnement distribué. Une autre perspective intéressante est d’implémenter ces deux processus sur plusieurs processeurs de cartes graphiques pour bénéficier de leur capacité de traitement massivement parallèle.

Dans nos travaux futurs, nous nous intéresserons aussi à comparer CPT avec plusieurs autres modèles de prédiction de séquences tels que *Context Tree Weighting (CTW)* et *Probabilistic Suffix Trees (PST)*. De plus, un nombre important d’améliorations ont été proposées dans la littérature pour les modèles présentés dans ce travail. À titre d’exemple, PPAM, PPM* et PPMC sont trois variantes de PPM qui promettent de meilleures performances que PPM [17, 28]. Des jeux de données additionnels représentant d’autres applications comme la recommandation de musique permettraient de solidifier notre plateforme d’évaluation.
Il serait également possible d’adapter CPT à la prédiction de séquences dans le contexte d’un flux infini de séquences. Dans de telles applications, la quantité d’information utilisée pour l’entraînement de modèles est considérée infinie et donc un mécanisme d’oubli est nécessaire. La gestion de tendances, nouvelles ou expirées, serait aussi indispensable [4].
ANNEXE A

WBPL : UNE LIBRAIRIE OPEN SOURCE POUR LA PRÉDICTION DE COMPORTEMENT SUR LE WEB

Abstract

We present WBPL (Web users Behavior Prediction Library), a cross-platform open-source library for predicting the behavior of web users. WBPL allows training prediction models from server logs. The proposed library offers support for three of the most used webservers (Apache, Nginx and Lighttpd). Models can then be used to predict the next resources fetched by users and can be updated with new logs efficiently. WBPL offers multiple state-of-the-art prediction models such as PPM, All-K-Order-Markov and DG, and a novel prediction model CPT (Compact Prediction Tree). Experiments on various web click-stream datasets shows that the library can be used to predict web surfing or buying behaviors with a very high overall accuracy (up to 38 %) and is very efficient (up to 1000 predictions /s).

1 Introduction

Sequences prediction algorithms have been widely integrated in webservers for numerous applications such as reducing latency by prefetching content, predicting buying behavior and recommending products [2, 18, 22]. The prediction task usually consists in predicting the next x resources or pages that a user will fetch using information contained in sequences from previous users. The prediction task is challenging since it is expected that prediction models can be trained efficiently and incrementally, and that fast and accurate predictions can be performed. Although there is an important need for predicting the next objects that a user will prefetch, there generally exists few public libraries that can be used for this task. Moreover, to our knowledge all existing libraries suffer from one or more of the following limitations. Several libraries offering sequence prediction models do not offer web-specific functionalities such as reading HTTP logs and filtering unwanted information [25]. Other libraries are specialized for application domains such as biological sequence prediction [29]. Therefore, these libraries cannot be directly used in the Web context.
without performing time-consuming modifications, and their efficiency has not been demonstrated for this task. Another important problem is that prediction libraries offer prediction models that cannot be trained incrementally [25].

In this paper, we address these limitations by proposing an open-source library for the prediction of web users behavior, named WBPL (Web users Behavior Prediction Library). The library offers implementation of state-of-the-art prediction models such as Prediction by Partial Matching [1], Dependency Graph [2], All-K-Order Markov [3] and our own approach; Compact Prediction Tree [21]. The library is cross-platform. Prediction models and other core components are implemented in Java. Furthermore, we provide a web GUI implemented in Javascript to quickly test and compare prediction models on user uploaded weblogs, using various parameters. The library can be easily integrated in web applications since it is open-source, has no external dependencies and provides important web-specific functionalities. Source code is available at http://goo.gl/Q1HLR0.

The rest of this paper is organized as follows. Section 2 explains how to use the GUI to quickly test prediction models and compare their performance. Section 3 explains how the library can be integrated in web applications to perform prediction. Finally, section 4 draws a conclusion.

2 Presented Models

The problem of sequence prediction. Given a finite alphabet of items \( I = \{i_1, i_2, ..., i_m\} \), an individual sequence is defined as \( S = \langle s_1, s_2, ..., s_n \rangle \), a list of ordered resources where \( s_i \in I \) (\( 1 \leq i \leq m \)). Let \( T = \{S_1, S_2, ..., S_t\} \) be a set of training sequences used to build a prediction model \( M \). The problem of sequence prediction consists in predicting the next item \( s_{n+1} \) of a given sequence \( \langle s_1, s_2, ..., s_n \rangle \) by using the prediction model \( M \).

Prediction by Partial Matching [1] (PPM) considers the last \( K \) items of a sequence to perform a prediction, where \( K \) is the order of the model. A \( K \)-order PPM model can be represented as a graph where nodes are subsequences and arcs indicate the probabilities that a subsequence will be followed by another subsequence. In a \( K \)-order PPM, arcs are outgoing from sequences of \( K \) consecutive items to nodes containing a single item. Predicting a sequence is performed by identifying the node containing the sequence’s last \( K \) items and then following the arc having the
highest probability. This approach has been proven to yield good results in certain areas such as web sequence predictions [1, 18]. However, this approach is very sensitive to noise. The smallest variation in a sequence will affect the prediction outcome. Because of this, prediction accuracy can greatly deteriorate for noisy datasets.

The All-$K$-Order Markov Model (AKOM) is a variation of the previous approach, consisting of training PPM predictors of order 1, 2...$K$ to perform predictions. AKOM yields higher accuracy than fixed order PPM models in most cases [18]. But it suffers from a much higher space complexity. A lot of research has been done to improve the speed and memory requirement, for example by pruning nodes [3, 18, 19], but few to improve accuracy.

The Dependency Graph (DG) [2] model is a graph where each node represents an item $i \in I$. A directional arc connects a node $A$ to a node $B$ if and only if $B$ appears within $x$ items from $A$ in training sequences, where $x$ is the lookahead window length. The weight of the arc is $P(B|A)/P(A)$ for the training sequences. Predicting a sequence consists of finding the node containing its last item and then following the arc with the highest weight.

Compact Prediction Tree (CPT) is our proposed prediction model [21]. It is built on the hypothesis that a lossless prediction model would yield higher accuracy than lossy models such as DG, PPM and AKOM because all relevant information from training sequences could be used to perform each prediction. The CPT consists of three data structures. The Prediction Tree (PT) is a tree-like data structure that stores and compresses training sequences. The Inverted Index (II) is a structure designed to efficiently identify in which sequences a set of items appears. Finally, the Lookup Table (LT) is an index that provides an efficient way to find a specific sequence in the prediction tree using its id. A CPT can be built and updated incrementally.

Predicting the next item of a sequence $S$ using a CPT is performed as follows. Let $x$ be an integer named the prefix length. The prediction algorithm finds all sequences containing the last $x$ items of $S$ in any order and in any position, using the inverted index. Next, these sequences are traversed in the prediction tree to count the number of occurrences of each item appearing after the last $x$ items of $S$. The item having the largest occurrence count is the predicted item. The prediction algorithm is tolerant to noise because it does not look for items appearing in a fixed
order. Furthermore, it incorporates a strategy named the *recursive divider* to remove potentially noisy items (cf. [21] for more details).

### 3 Using the Web GUI

In this section, we describe the four steps to use the web GUI. The web GUI (cf. Fig. A-1) is useful to compare prediction models on web log data with various parameters, to evaluate their accuracy for a given web application.

![WSBL user interface](image)

**Figure A-1** WSBL user interface

**Step 1: Choosing a data source.** The user should first upload its own raw log file in the default format of one of the three main supported webserver platforms (Nginx, Lighttpd or Apache), or select one of three well known public web datasets namely BMS, FIFA, and Kosarak. BMS contains 15,806 sequences (average length of 6 items) and 495 unique items. FIFA contains 28,978 sequences (average length of 32 items) and 3,301 unique items. Kosarak contains 638,811 sequences (average length of 32 items) and 3,301 unique items. Kosarak contains 638,811 se-
quences (average length of 12 items) and 39,998 unique items. Uploaded log files are converted into sequences of items that can be used by the prediction models. Converting a log file into a set of sequences is done as follows. Each sequence represents a web session of a user. A web session is a sequence of HTTP requests found in the logs. A user is identified by its IP address and user-agent if it is provided. Two consecutive HTTP requests from a user belong to the same web session if the time difference between the two is smaller than the time window. The time window is a parameter that must be specified in the user interface. Note that multiple individual sessions for a given user are not grouped together because of the limited information provided in log files. For more accurate identification of users, a webserver could use a user authentication mechanism or a permanent tracking cookie.

**Step 2. Choosing prediction model(s).** The user next has to select one or more prediction models for comparison. Each model has its own set of parameters that needs to be set (cf. Section 2 for a description of each parameter). All the predictors do not perform equally well in all situations. Therefore, it is recommended to test them all for a given dataset.

**Step 3. Tuning parameters for running an experiment.** The user should next select parameters to indicate how to compare the selected prediction models. These parameters are independent of the choice of prediction models. The \( \text{MinS} \) parameter indicates the minimum sequence length. Any sequence shorter than \( \text{MinS} \) will not be used for the experiment (either for training or testing). Setting the right value for this parameter may enhance the accuracy of prediction models because some sequences are too short to generate accurate predictions. Note that PPM and DG are not affected by this parameter. The user should also select a sampling strategy to determine how to divide the dataset into training and test sets. Two sampling methods are offered. The *standard random sampling* randomly split the data into two sets, training and testing, by specifying the relative ratio (in the range \([0, 1]\)) of training data desired. The *K-Fold cross validation* performs a standard k-fold cross-validation. It is slower than the standard random sampling but offers more reliable results. Finally, the user should set two parameters with respect to test sequences. The *suffix size* indicates the number of items of a test sequence to be used for verifying a prediction, while the *prefix size* indicates the number of items to be used for making the prediction. For example, for a sequence \( \langle s_1, s_2, s_3, s_4, s_5 \rangle \), suffix size = 1 and prefix size = 2, the items \( s_3 \) and \( s_4 \) will be used to make the
prediction and \( s_5 \) will be used to verify the prediction. If the predicted item appears in the suffix (here, \( s_5 \)), the prediction is accurate, otherwise not. Choosing a larger suffix size generally increases prediction accuracy. Choosing a larger prefix size may also increase accuracy, except for PPM and DG.

**Step 4. Running the experiment.** The user should next click on the "Start Experiment" button to launch the experiment. The library loads and converts the raw log file into a set of sequences if one of the provided dataset is not selected. Once converted, the dataset is split into a training set and a testing set according to the sampling strategy. Each selected prediction model is then trained with the training set by processing one training sequence at a time. Testing sequences are then used to evaluate accuracy, coverage and time of each prediction model.

**Experimental results.** To evaluate the performance of the library, we compared the prediction models on the BMS dataset. Models were tuned with their best parameters (DG lookahead window = 4, AKOM order = 5). We ran the experiment on a 3rd generation Core i3 processor running Windows 7. We found that the library can predict next fetched resources with a high accuracy. For BMS the most and second most accurate models were CPT (38.4 \%) and DG (36 \%). Results also show that the library is very efficient. For BMS the best and second best models in terms of predictions per second are PPM (1000 /s) and DG (250 /s). We also found that the library is very efficient in terms of training time. The fastest and second fastest models for inserting a sequence for BMS are PPM (0.01 s) and CPT (0.02 s). The result section of the GUI presents, as charts A-2, the performances of each selected model in terms of accuracy, spatial size and time.

4 Using the Library

WBPL is composed of two parts, a web GUI and a library. The web GUI consists of a small HTML, CSS and Javascript website. The interface can be uploaded to a server and linked to the backend framework using the provided PHP bridge. Since running experiments using the web GUI can be resource intensive, we recommend using a dedicated environment for testing purposes. The library is written in Java and the source code is available at [http://goo.gl/Q1HLR0](http://goo.gl/Q1HLR0). The source code is highly modular and easy to customize for specific needs. The library provides an
interface to add new prediction models or edit existing ones. Just like when using the web GUI, it is possible to load custom log files such as Apache logs, Nginx logs or Lighttpd logs, and use them as datasets. The library is designed to be used alongside a web application. In this case, the application has to first train the desired prediction model(s) with training data and then the application can ask the library to perform sequence predictions. We provide a simple PHP bridge to communicate back and forth between the library and a web application.

5 Conclusion

We presented WBPL, a cross-platform open-source library for predicting the next resources fetched by web users. The library offers implementations of state-of-the-art prediction models DG, PPM, AKOM and CPT. Furthermore, if offers support for reading logs files from the three main webservers (Apache, Nginx and Lighttpd). We also presented a web interface to quickly test the prediction models offered in the library with various parameters on user-given web datasets. The web interface provides an intuitive and non-intrusive way for users to experiment with the prediction models before committing to one. Experiments on real-life web click-stream datasets shows that the library can achieve a very high overall accuracy (up to 38 % on the presented datasets) and can perform very fast prediction (up to 1000 predictions /s). For future work, we will work on
further enhancing the performances of the proposed prediction models [3, 18, 19], proposing additional models such as Context Tree Weighting and Neural Networks, and providing ways to use the library with other web frameworks like Ruby on Rails and .NET.
RÉFÉRENCES


