C.CPT
A succinct representation of Compact Prediction Tree

Rafael Ktistakis
crk15@le.ac.uk
Supervised by: Prof. Rajeev Raman
Department of Computer Science
University of Leicester

January 2017

Contents

List of Figures

List of Tables

1 Introduction

Predicting the next item of a sequence over a finite alphabet has various approaches where some of them contain a relatively extensive literature. The applications which are covered by these various prediction approaches vary and different challenges are addressed for each one. *Lossless* approach is a new approach recently introduced by Gueniche et al. 2013. Since it is a new approach, it is interesting to study it offering more solutions.

2 Problem Definition

Let a finite set of items (symbols) $\Sigma = \{e_1, e_2, \ldots, e_\sigma\}$ be called *alphabet*, then a *sequence* S is a list of ordered items $S = \langle i_1, i_2, \ldots, i_k \rangle$ where $i_m \in \Sigma \; for \; 1 \leq m \leq k$. A multiset of sequences $D = \langle S_1, S_2, \ldots, S_l \rangle$ also called *dataset* is constituted by sequences that created under a similar way. Given a dataset, a prediction algorithm is *trained* on this dataset. Once the prediction algorithm is trained, it will repeatedly accept a *context* which is another sequence of symbols, and outputs a *prediction*. The context is obtained from an unknown *query sequence* which does not belong to the dataset but shares common characteristics with the dataset. Loosely speaking, the context is derived from a part of the query sequence, and the prediction is about what comes *after* the context in the query.

3 Lossless

Lossless sequence prediction is a relatively new approach and therefore appears to have quite a few challenges that someone that studies it has to cope with. It is clear that due to the fact that a whole dataset is utilised, a lossless approach achieves a higher space utilisation over a lossy approach. This fact leads to two results; firstly, it is somewhat intensive for an algorithm to go through all the data and extract the appropriate information needed for predictions, and secondly, keeping all the relative data *in memory*, through the corresponding data structures, appears to be a considerably hard challenge because the represented structure's space utilisation often reaches an order of magnitude of the initial gathered data. As a result, a lossless sequence prediction approach needs sophisticated tools to search for patterns and sequences in *Big Data* when simultaneously the aforementioned challenges are coped.

On contrary, a *lossy* approach can be less memory hungry by representing the given information of a dataset through models (like *dependency graphs*). This leads to implementation that sometimes utilises less space and can be potentially faster. However, a lossy approach lacks accuracy especially in hard cases when information in a dataset is not repeating as other parts of informations or a repetition is either generalised or ignored by a model having a partly loss of information that is not reversible. Often such hard cases lead to more complex models that along with Big data is hard to make algorithms that produce predictions with efficiency and performance.

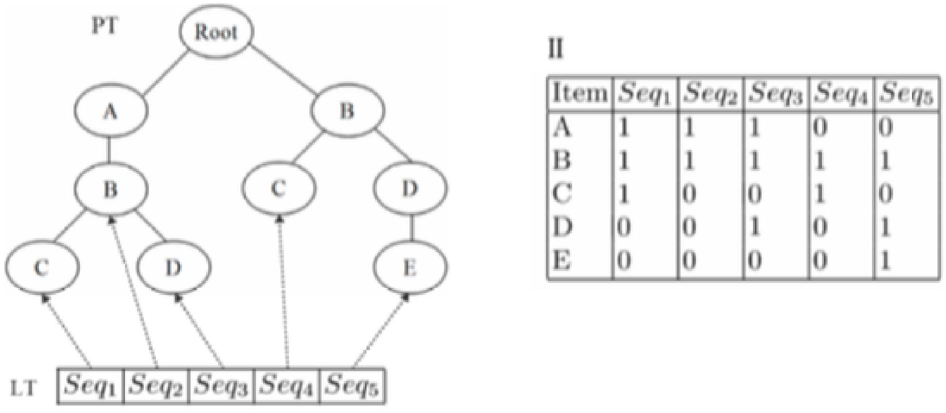
4 Motivation

refer briefly to some SPICE results

5 CPT Predictor and its Data Structures

CPT and CPT+ (**C**ompact **P**rediction **T**ree) introduced by Gueniche et al. 2013 and later improved by Gueniche et al. 2015, is currently the only implementation in lossless sequence prediction. It is constituted by a Trie (*Prediction Tree PT*), an Array of bit-vectors (*Inverted Index II*) and an array of pointers to the last items of sequences inserted to the prediction tree (*Look-up Table LT*). As a lossless approach, CPT utilises the whole dataset provided for building its structures. Each sequence from the dataset is inserted to the Prediction Tree and a pointer to the last item of the sequence is added to the Look-up Table. The Inverted Index has the role of finding which sequence contains which alphabet's items by simply executing a bitwise AND operation. Every bit-vector has the same length with the size of the dataset's alphabet while it notes, using 1/0, whether an alphabet item appears in the sequence or not.

CPT can retrieve any searching sequence with specific alphabet items by using the inverted index along with a combination of the two other structures. A bitwise AND shows which sequences contain some given alphabet items and then these sequences can be located by using the Look-up Table which points to the end of each corresponding sequence. Predictor task - retrieving consequent



Item	Seq_1	Seq_2	Seq_3	Seq_4	Seq_5
A	1	1	1	0	0
B	1	1	1	1	1
C	1	0	0	1	0
D	0	0	1	0	1
E	0	0	0	0	1

Figure 1: A Prediction Tree (PT), Inverted Index (II) and Lookup Table (LT)

In Figure 1 it is shown an example of CPT during multiple sequence insertions.

5.1 Strengths & Weaknesses

CPT (as a lossless approach) utilises all the available sequences in a tree where they are accessible through the II and the LT easily and fast by a bitwise AND for each sequence. Therefore, making predictions can be potentially fast and accurate as described in [2] and [3]. Locating similar sequences can be a relatively easy task that makes CPT work as a good tool for predictors that exploit such sequences. Moreover, CPT can implement all the prediction tasks described in Section C, offering a bigger diversity for implementations of several applications.

However, even though with the improvement of CPT, where size of the structures were reduced and speed was enhanced, some performance issues can be still addressed. Bit-vectors are the main element of II which is used for locating CPT's sequences. When more and more sequences are being added, the bitwise AND operations are becoming slower and slower, leading to performance issues and overall scalability issues since the vectors are becoming much larger. It is common for real life applications to deal with datasets that contain an enormous number of sequences. Therefore, Bit-vectors can be CPT's Achilles' heel regarding its peformance. Also, space usage of II in such cases can be wasteful since with Succinct Data Structures (see Section A) a better and more memory efficient structure can be proposed.

5.2 Prediction Tree

operations used by the predictor

5.3 Inverted Index

operations used by the predictor

5.4 Look-Up Table

operations used by the predictor

6 Succinct CPT

6.1 SD-Vectors as a Prediction Tree

6.2 Elias-Fano for Inverted indicies

6.3 Look-Up Table ommition

7 Experimental Results and Performance Comparison

7.1 Space Comparison

7.2 Speed Comparison

A Succinct Data Stuctures

This kind of structures are high space efficient data structures (introduced by Jacobson 1989) that support rapid and efficient operations, (look Appendix B), like *rank & select* [1, 4]. A data structure in order to be considered *succinct*, it should use space that approaches the information-theoretic lower bound of the space that is required to represent the data. One interesting fact is that in contrast with other compressed representations succinct data structures do not sacrifice performance in order to deliver space efficiency when represent and retrieve back any relative data.

A.0.1 Why Succinct Data Structures

Using *Succinct data structures* can help reducing the space required by a lossless approach and keep most of the data "in memory" (which is stated as a main challenge in Section **??**). This leaves open room to improve an algorithm's scalability. If we consider that compression methods are mostly based in compression and decompression mechanisms, it is almost clear that much time is spent in compressing and decompressing data. Even though, it would have been achieved a space reduction by using a compression method, the overall performance would not have been any good. Even worse, in a scalable level such an algorithm would take huge amounts of time to complete leaving out an important aspect of the algorithm which is scalability. Also, if we consider prediction tools like backward search on FM-Index (Section **??**) which their complexity to find a search pattern depends only on the times a rank/select were performed, this leads to algorithms which are more scalable with a robust performance time which is in-dependable of the input training set; aspect crucial for an algorithms performance.

B Functionality of Rank & Select

Let B be a sequence of n items. Then **rank(c)** and **select(c)** can be defined as:

rank(o) Counts the number of items which appear in B from its start to its o-th position [4].

select(o) Finds the o-th occurrence of an element in B [4].

The time complexity of rank/select queries depends on the structure where these two operation are built on. For example, it is possible to execute rank/select in constant $\mathcal{O}(1)$ time for some bit-vectors, like rrr-vectors [5], or another binary rank index [1].

C Prediction Tasks

The adoption of which item is predicted after the context (basically the value of w) is only specified by the prediction task. During this report, there are four different prediction tasks that are mentioned and used; Right next item, Item in a future window, Top K predictions and finally Distribution of right next items.

Right next item: The prediction algorithm gives one prediction for the q_{j+1} coming after the given context.

Top K predictions: The prediction algorithm gives K alternative predictions for the q_{j+1} item of the given context.

Distribution of right next items: For each item of the alphabet Σ, the prediction task gives the item's probability to be the q_{j+1}-th item.

Item in a future window For a given value of w, it is predicted an item that appears somewhere among the q_{j+1}-th, q_{j+2}-th, $\ldots$, q_{j+w}-th items. So, w mostly behaves as a *window* value where the predicted item could appear.

References

[1] Craig Dillabaugh. *Succinct Data Structures*. 2007 (cit. on p. 7).

[2] Ted Gueniche, Philippe Fournier-Viger, and Vincent S. Tseng. “Compact Prediction Tree: A Lossless Model for Accurate Sequence Prediction”. In: *Advanced Data Mining and Applications* (2013), pp. 177–188. DOI: `10.1007/978-3-642-53917-6_16` (cit. on pp. 4, 5).

[3] Ted Gueniche, Philippe Fournier-Viger, Rajeev Raman, and Vincent S. Tseng. “CPT+: Decreasing the Time/Space Complexity of the Compact Prediction Tree”. In: *Lecture Notes in Computer Science* (2015), pp. 625–636. DOI: `10.1007/978-3-319-18032-8_49` (cit. on pp. 4, 5).

[4] Guy Jacobson. “Space-efficient Static Trees and Graphs”. In: *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*. 1989, pp. 549–554. DOI: `10.1109/SFCS.1989.63533`. URL: `http://dx.doi.org/10.1109/SFCS.1989.63533` (cit. on p. 7).

[5] Rajeev Raman, Venkatesh Raman, and S. Srinivasa Rao. “Succinct Indexable Dictionaries with Applications to Encoding K-ary Trees and Multisets”. In: *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA ’02. San Francisco, California: Society for Industrial and Applied Mathematics, 2002, pp. 233–242. ISBN: 0-89871-513-X. URL: `http://dl.acm.org/citation.cfm?id=545381.545411` (cit. on p. 7).