European Journal of Operational Research 000 (2016) 1–10

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Innovative Applications of O.R.

An approximate dynamic programming approach for improving accuracy of lossy data compression by Bloom filters

Xinan Yang[a,*], Alexei Vernitski[a], Laura Carrea[b,c]

[a] Department of Mathematical Sciences, University of Essex, Wivenhoe Park, Colchester CO4 3SQ, UK
[b] Department of Meteorology, University of Reading, Reading RG6 6BB, UK
[c] School of Computer Science and Electronic Engineering, University of Essex, Wivenhoe Park Colchester CO4 3SQ, UK

ARTICLE INFO

Article history:
Received 27 January 2015
Accepted 20 January 2016
Available online xxx

Keywords:
Lossy compression
Bloom filter
Integer linear program
Approximate dynamic programming
Heuristics

ABSTRACT

Bloom filters are a data structure for storing data in a compressed form. They offer excellent space and time efficiency at the cost of some loss of accuracy (so-called lossy compression). This work presents a yes–no Bloom filter, which as a data structure consisting of two parts: the yes-filter which is a standard Bloom filter and the no-filter which is another Bloom filter whose purpose is to represent those objects that were recognized incorrectly by the yes-filter (that is, to recognize the false positives of the yes-filter). By querying the no-filter after an object has been recognized by the yes-filter, we get a chance of rejecting it, which improves the accuracy of data recognition in comparison with the standard Bloom filter of the same total length. A further increase in accuracy is possible if one chooses objects to include in the no-filter so that the no-filter recognizes as many as possible false positives but no true positives, thus producing the most accurate yes–no Bloom filter among all yes–no Bloom filters. This paper studies how optimization techniques can be used to maximize the number of false positives recognized by the no-filter, with the constraint being that it should recognize no true positives. To achieve this aim, an Integer Linear Program (ILP) is proposed for the optimal selection of false positives. In practice the problem size is normally large leading to intractable optimal solution. Considering the similarity of the ILP with the Multidimensional Knapsack Problem, an Approximate Dynamic Programming (ADP) model is developed making use of a reduced ILP for the value function approximation. Numerical results show the ADP model works best comparing with a number of heuristics as well as the CPLEX built-in solver (B&B), and this is what can be recommended for use in yes–no Bloom filters. In a wider context of the study of lossy compression algorithms, our research is an example showing how the arsenal of optimization methods can be applied to improving the accuracy of compressed data.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

In information technology, lossy compression is a data compression method that reduces the size of the representation at the cost of the loss of some accuracy at the decompression time. In exchange for losing accuracy in representation, lossy data structures not only store all information in constant space but also respond to membership queries in constant time. Examples of lossy data structures include skip lists (Pugh, 1989), lossy dictionaries (Pagh & Rodler, 2001) and several hashing techniques.

1.1. Bloom filter

The Bloom filter is one of lossy methods of storing compressed data, introduced in Bloom (1970). The kind of data that Bloom filter is especially suitable for are sets. Given a set, a Bloom filter can be produced which represents the set in a compressed form. It can then be queried in the sense that there is an algorithm which, given an object and a Bloom filter representing a set, decides whether the object is or is not an element of the set. The querying algorithm is very efficient and works extremely fast compared to standard algorithms of accessing compressed data (one of the reasons why this algorithm is fast is that it contains many operations which are performed in parallel and that it is easy to implement in hardware) (Tarkoma, Rothenberg, & Lagerspetz, 2012a). The size of the Bloom filter can be very small compared to standard ways of compressing data, which is a major advantage of Bloom filters. Nevertheless, there is also an important disadvantage: Bloom

* Corresponding author. Tel.: +01206 872787.
E-mail address: xyangk@essex.ac.uk, sibylnan@163.com (X. Yang).

http://dx.doi.org/10.1016/j.ejor.2016.01.042
0377-2217/© 2016 Elsevier B.V. All rights reserved.

filters only represent data approximately, and frequently the querying algorithm gives an incorrect answer to the question about the membership of an object in the set represented by a Bloom filter.

The broad area of applicability of Bloom filters, due to their excellent space and time efficiency, is either in low-performance hardware or for tasks which must be performed extremely fast and speed is slightly more important than accuracy. Bloom filters have a range of uses in information technology (Broder & Mitzenmacher, 2002; Tarkoma, Rothenbergand, & Lagerspetz, 2012b), from hardware implementations to software applications domain, where it was first conceived to perform space and time efficient dictionary lookups (Bloom, 1970). Broder and Mitzenmacher (2002) have coined the Bloom filter principle: 'Whenever a list or set is used, and space is at a premium, consider using a Bloom filter if the effect of false positives can be mitigated'. To give just one example, Bloom filters can be used for routing in computer networks: in this application, a path which a message must follow is represented by a Bloom filter, namely, as a union of those links between computers which together constitute the path. It is appropriate to use Bloom filters in this application because each computer along the path must decide where to forward the message very quickly (literally with the speed of light, assuming that the links between computers are optical cables).

Calculating a Bloom filter of one object is a preliminary stage before building or querying a Bloom filter representing a set of objects. Assume that there is an algorithm which takes any object as its input and produces a binary array of a fixed length G, in which H bits are equal to 1 and other bits are equal to 0. We refer to this array as the Bloom filter of the object. The purpose of the Bloom filters of objects is to serve as uniform labels for each object which may interest us. Informally speaking, the Bloom filters of objects may be likened to bar-codes glued to every object. We denote the Bloom filter of an object s by $\eta(s)$.

Given a set $\mathcal{S}$, the Bloom filter of $\mathcal{S}$ can be computed as the bitwise disjunction of the Bloom filters of the elements in $\mathcal{S}$; in other words, the Bloom filter of the set $\mathcal{S}$ is defined as a binary array of length G, and for each $j = 1, \ldots, G$ the jth bit is calculated as follows: if in every $\eta(s)$, where $s \in \mathcal{S}$, the jth bit is 0 then the jth bit of the Bloom filter is 0; otherwise, if in at least one $\eta(s)$ the jth bit is 1 then the jth bit of the Bloom filter is 1. We denote the Bloom filter of a set $\mathcal{S}$ by $\beta(\mathcal{S})$.

Given an object s and a Bloom filter $\beta(\mathcal{S})$, querying it to determine whether s is an element of $\mathcal{S}$ is done as follows. For each $j = 1, \ldots, G$ if the jth bit of $\eta(s)$ is less than or equal to the jth bit of $\beta(\mathcal{S})$ then we say that s is recognized[1] by the Bloom filter $\beta(\mathcal{S})$ as belonging to the set $\mathcal{S}$. In an ideal world, one would like to be able to claim that s is recognized as belonging to $\mathcal{S}$ if and only if s is an element of $\mathcal{S}$. However, this is not so. Due to the definition of a Bloom filter, if s is an element of $\mathcal{S}$ then s is recognized by $\beta(\mathcal{S})$; but the converse is not true: not necessarily an object s recognized by $\beta(\mathcal{S})$ is an element of $\mathcal{S}$. This kind of error is called false positives, in the sense that the Bloom filter query recognizes the element as belonging to the set, but should not do it.

A number of approaches have been proposed to reduce the number of false positives in Bloom filters. The number H of positions equal to 1 can be varied (Kirsch & Mitzenmacher, 2008). Generalizations of the standard Bloom filter have also been considered, such as the yes–no Bloom filter (Vernitski, Carrea, & Reed, 2015) that is further studied in this paper, the retouched Bloom filter (Donnet, Baynat, & Friedman, 2006), the counting Bloom filter (Guo, Liu, Li, & Yang, 2010), the power of two choices (Lumetta & Mitzenmacher, 2007), the optihash (Carrea, Vernitski, & Reed, 2014) or partitioned hashing (Hao, Kodialam, & Lakshman, 2007). Both the standard Bloom filter and its generalizations listed above can work in the use scenario in which some false positives and some false negatives are allowed. Nevertheless, in this paper we concentrate on the construction of yes–no Bloom filter under the standard use scenario in which we allow some false positives (trying to minimize their number) and do not allow any false negatives (for example, in the application to routing this approach means that the message will definitely be delivered to the right recipient but perhaps also sent to some other computers, thus creating some unnecessary traffic in the network).

1.2. Yes–no Bloom filter

This paper studies a yes–no Bloom filter (Vernitski et al., 2015), which is our new generalization of the standard Bloom filter which actively reduces the number of false positives at the stage of building the Bloom filter. Let us start with a fictitious and simplified but realistic use scenario. Suppose the management of an airport installs CCTV cameras whose output is automatically compared with photographs of 100 known terrorist suspects. Suppose these photographs are stored in a compressed form as a Bloom filter. Due to Bloom filters' efficiency, even low-performance hardware can effectively compare faces of people in the airport with the faces of the suspects. Then, if the Bloom filter recognizes a face, the security staff is called to look at the person and make a decision. As we have discussed, Bloom filters produce false positives; therefore, the security staff will be called more often than needed. In particular, suppose that out of all the employees working in the airport, 300 people have faces that trigger false positives. We can considerably reduce unnecessary checks and nuisance if we actively indicate to the Bloom filter that these specific objects are recognized incorrectly and should not be recognized.

A yes–no Bloom filter consists of two Bloom filters, one called a yes-filter and the other called a no-filter.[2]

Now we shall define the algorithms for building the yes–no Bloom filter of a set and for querying a yes–no Bloom filter. We assume that each object has two Bloom filters corresponding to it: the *yes-filter* of length G^+ and the *no-filter* of length G^-. Given an element s, we shall denote its yes-filter by $\eta^+(s)$ and its no-filter by $\eta^-(s)$.

Consider a set $\mathcal{S}$ and another set $\mathcal{T}$ such that that the two sets do not overlap. The set $\mathcal{S}$ is the one that we want to store in a compressed form, and the set $\mathcal{T}$ is a set whose elements are likely to be queried but should not be recognized as elements of $\mathcal{S}$. (In the example above, $\mathcal{S}$ is the set of suspects and $\mathcal{T}$ is the set of airport employees.) Build the yes-filter $\beta^+(\mathcal{S})$ of $\mathcal{S}$ as the bitwise disjunction of all $\eta^+(s)$, where $s \in \mathcal{S}$; in other words, $\beta^+(\mathcal{S})$ is the standard Bloom filter built from all arrays $\eta^+(s)$. The Bloom filter $\beta^+(\mathcal{S})$ will have false positives, including some which are contained in $\mathcal{T}$; let us denote the subset of $\mathcal{T}$ consisting of false positives of $\beta^+(\mathcal{S})$ by $\mathcal{F}$. (In the example above, $\mathcal{F}$ is the set of those employees whose faces are recognized by the Bloom filter.) Then the second, more interesting stage begins: we build the no-filter $\beta^-(\mathcal{S})$ of $\mathcal{S}$ so that $\beta^-(\mathcal{S})$ recognizes as many as possible elements of $\mathcal{F}$ but none of the elements of $\mathcal{S}$. (As we shall see later in the paper, unlike building a standard Bloom filter or the yes-filter, this stage involves flexibility as to which elements of $\mathcal{F}$ should be included in $\beta^-(\mathcal{S})$, and, therefore, turns into a meaningful optimization problem.)

[1] It is also convenient to use the pure-mathematics term 'covered', that is, s is covered by $\beta(\mathcal{S})$, thus stressing that $\beta(\mathcal{S})$ is a lattice join (or, in another interpretation, a set union) of the Bloom filters of the elements of $\mathcal{S}$.

[2] Slightly more general approaches are also possible, which involve more than two Bloom filters, but this particular choice in the design of the structure is considered for the purposes of this paper. See the conclusion for suggestions of further research.

Download English Version:

https://daneshyari.com/en/article/6895682

Download Persian Version:

https://daneshyari.com/article/6895682

Daneshyari.com