Compressed Perfect Embedded Skip Lists for Quick Inverted-Index Lookups

Paolo Boldi and Sebastiano Vigna

Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano

Abstract. Large inverted indices are by now common in the construction of web-scale search engines. For faster access, inverted indices are indexed internally so that it is possible to skip quickly over unnecessary documents. To this purpose, we describe how to embed efficiently a *compressed perfect skip list* in an inverted list.

1 Introduction

The birth of web search engines has brought new challenges to traditional inverted index techniques. In particular, *eager* (or *term-at-a-time*) query evaluation has been replaced by *lazy* (or *document-at-a-time*) query evaluation. In the first case, the inverted list of one of the terms of the query is computed first (usually, choosing the rarest term [4]), and then, it is merged or filtered with the other lists. When evaluation is lazy, instead, inverted lists are scanned in parallel, retrieving in sequence each document satisfying the query.

Lazy evaluation requires keeping constantly in sync several inverted lists. To perform this operation efficiently, it is essential that a *skip* method is available that allows the caller to quickly reach the first document pointer larger than or equal to a given one. The classical solution to this problem [1] is that of embedding skipping information in the inverted list itself: at regular intervals, some additional information describe a *skip*, that is, a pair given by a document pointer and the number of bits that must be skipped to reach that pointer. The analysis of skipping given in [1] concludes that skips should be spaced as the square root of the term frequency, and that one level of skip is sufficient for most purposes.

Nonetheless, the abovementioned analysis has two important limitations. First of all, it does not contemplate the presence of *positions*, that is, of a description of the exact position of each occurrence of a term in a document, or of application-dependent additional data, thus underestimating the cost of *not* skipping a document; second, it is fundamentally based on eager evaluation, and its conclusions cannot be extended to lazy evaluation. Motivated by these reasons, we are going to present a generic method to self-index inverted lists with a very fine level of granularity. Our method does not make any assumption on the structure of a document record, or on the usage pattern of the inverted index. Skips to a given pointer (or by a given amount of pointers) can always be performed with a logarithmic number of low-level reads and bit-level skips: nontheless, the size of the index grows just by a few percents.

Our techniques are particularly useful for *in-memory indices*, that is, for indices kept in core memory (as it happens, for instance, in Google), where most of the computational cost of retrieving document is scanning and decoding inverted lists (as opposed to disk access), and at the same time a good compression ratio is essential. All results described in this paper have been implemented in MG4J, available at `http://mg4j.dsi.unimi.it/`.

2 Perfect Embedded Skip Lists for Inverted Indices

Perfect skip lists. Skip lists [3] are a data structure in which elements are organised as in an ordered list, but with additional references that allow one to skip forward in the list as needed. More precisely, a *skip list* is a singly linked list of increasingly ordered items x_0, x_1, … , x_{n-1} such that every item x_i, besides a reference to the next item, contains a certain number $h_i \geq 0$ of extra references, that are called the *skip tower* of the item; the t-th reference in this tower addresses the first item $j > i$ such that $h_j \geq t$.

We are now going to describe *perfect skip lists*, a deterministic version of skip lists (which were originally formulated in randomised terms) that is suitable for inverted lists. We fix two limiting parameters: the number of items in the list, and the maximum height of a tower.

For sake of simplicity, we start by describing an ideal, infinite version of a perfect skip list. Let $\mathrm{LSB}(x)$ be defined as the least significant bit of x if x is a positive integer, and ∞ if $x = 0$. Then, define the height of the skip tower of item x_i in an infinite perfect skip list as $h_i = \mathrm{LSB}(i)$. We say that a finite skip list is *perfect* w.r.t a given height h and size T when no tower contains more than $h + 1$ references, and all references that would exist in an infinite perfect skip list are present, provided that they refer to an item with index smaller than or equal to T, and that they do not violate the first requirement.

Theorem 1. *In a perfect skip list with T items and maximum height h, the height of a tower at element i is* $\min(h, \mathrm{LSB}(k), \mathrm{MSB}(T - i)) + 1$*, where* $k = i \bmod 2^h$*, and* $\mathrm{MSB}(x)$ *is the most significant bit of* $x > 0$*, or* -1 *if* $x = 0$*. In particular, if* $i < T - T \bmod 2^h$ *the tower has height* $\min(h, \mathrm{LSB}(k)) + 1$*, whereas if* $i \geq T - T \bmod 2^h$ *the tower has height* $\min(\mathrm{LSB}(k), \mathrm{MSB}(T \bmod 2^h - k)) + 1$.

Addressing directly all pointers in an inverted list would create unmanageable indices. Thus, we shall index only one item out of q, where q is a fixed *quantum* that represents the minimally addressable block of items. Figure 1 shows a perfect skip list.

Embedding Skip Lists into Inverted Indices

The first problem that we have to deal with when trying to embed skip lists into an inverted index is that we want to access data in a strictly sequential manner, so the search algorithm we described cannot be adopted directly: we must store not only the bit offset of the referenced item, but also the pointer contained therein.

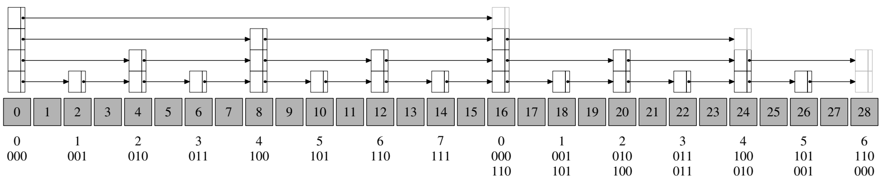

Fig. 1. A perfect skip list with $T = 31$ items, $q = 2$ and $h = 3$. The first line show the values of k; the second line their h-bit binary expansion. The list of made of two *blocks*, and for the latter (a *defective* block of $L = 15$ items), also the binary expansion of $\lfloor L/q \rfloor - k$ is shown. The ghosted references do not exist (they are truncated by minimisation with $\mathrm{MSB}(\lfloor L/q \rfloor - k)$).

Pointer Skips. Let us consider a document collection of N documents, where each term t appears with relative frequency p_t; according to the *Bernoulli model* [4], every term t is considered to appear independently in each document with probability p_t. As a result, the random variable describing a pointer skip to ℓ documents farther is approximated by a normal distribution with mean ℓ/p_t and standard deviation $\sqrt{\ell(1-p_t)}/p_t$ (details appear in the full paper). We also suggest to predict the pointer skips that are not a tower top using a *halving model*, in which a pointer skip of level s is stored as the difference from the skip of level $s+1$ that contains it, divided by 2 (standard deviation drops to $\sqrt{\ell(1-p_t)/2}/p_t$).

Gaussian Golomb Codes. We are now left with the problem of coding integers normally distributed around 0. We compute approximately the best Golomb code for a given normal distribution. This is not, of course, an optimal code for the distribution, but for $\sigma \gg 1$ it is excellent, and the Golomb modulus can be approximated easily in closed form: integers distributed with standard deviation σ are Golomb-coded optimally with modulus $2 \ln 2 \sqrt{\frac{2}{\pi}}\sigma \approx 1.106\,\sigma$.

Bit Skips. The strategy we follow for bit skips is absolutely analogous to that of pointers, but with an important difference: it is very difficult to model correctly the distribution of bit skips. We suggest a prediction scheme based on the average bit length of a quantum, coupled with a universal code such that γ or δ.

3 Inherited Towers

As remarked in the previous sections, the part of a tower that has greater variance (and thus is more difficult to compress) is the tower top. However this might seem strange, our next goal is to avoid writing tower tops at all. When scanning sequentially an inverted list, it is possible to maintain an *inherited tower* that represent all "skip knowledge" gathered so far (see Figure 2), similarly to *search fingers* [2].

Note that inherited entries might not reach height h if the block is defective (see the right half of Figure 1). Supposing without loss of generality that $q = 1$, we have:

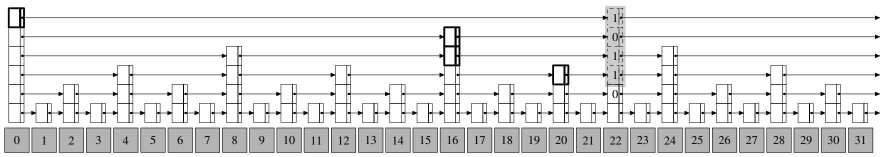

Fig. 2. Scanning the list ($q = 1$, $h = 5$), we are currently positioned on element $x = 22$, with a tower of height 2; its inherited tower is represented in grey, and the items it is inheriting from are in bold

Theorem 2. *The highest valid entry in an inherited tower for a defective block of length L is* $\mathrm{MSB}(L \oplus k)$*, where* $\oplus$ *denotes bit-by-bit exclusive or.*

The above computation leads us the following, fundamental observation: *a non-truncated tower with highest entry $\bar{h}$ inherits an entry of level $\bar{h} + 1$ that is identical to its top entry.* As a consequence, if lists are traversed from their beginning, top entries of non-truncated towers can be omitted. The omission of top entries halves the number of entries written, and, as we observed at the start of the paragraph, reduces even further the skip structure size.

Experimental Data and Conclusions

We gathered statistics indexing a partial snapshot of 13 Mpages taken from the `.uk` domain containing about 50GiB of parsed text (the index contained counts and occurrences). The document distribution in the snapshot was highly skewed, as documents appeared in crawl order. Adding an embedded perfect skip list structure with arbitrary tall towers caused an increase in size of 2.3% (317 MiB) with $q = 32$ and 1.23% when $q = 64$; indexing using the square-root spaced approach caused an increase of 0.85%. Compressing the same skip structures using a γ or δ code instead of Gaussian Golomb codes for pointer skips caused an increase in pointer-skip size of 42% and 18.2%, respectively.

Speed is, of course, at the core of our interests. The bookkeeping overhead of skip lists increases by no more than 5% (and by .5% on the average) the time required to perform a linear scan. On the contrary, tests performed on synthetically generated queries in disjunctive normal form show an increase in speed between 20 and 300% w.r.t. the square-root spaced approach.

References

1. Alistair Moffat and Justin Zobel. Self-indexing inverted files for fast text retrieval. *ACM Trans. Inf. Syst.*, 14(4):349–379, 1996.
2. William Pugh. A skip list cookbook. Technical report UMIACS-TR-89-72.1, Univ. of Maryland Institute for Advanced Computer Studies, College Park, College Park, MD, USA, 1990.
3. William Pugh. Skip lists: a probabilistic alternative to balanced trees. *Commun. ACM*, 33(6):668–676, 1990.
4. Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images.* Morgan Kaufmann Publishers, Los Altos, CA 94022, USA, second edition, 1999.