A Compressed-Gap Data-Aware Measure for Indexable Dictionaries

Nicola Prezza

Department of Mathematics and Computer Science, University of Udine, Italy
prezza.nicola@spes.uniud.it

Abstract. We consider the problem of building a compressed fully-indexable dictionary over a set $S$ of $n$ items out of a universe $U = \{0, ..., u - 1\}$. We use gap-encoding combined with entropy compression in order to reduce the space of our structures. Let $H_{\text{gap}}^0$ be the zero-order empirical entropy of the gap stream. We observe that $nH_{\text{gap}}^0 \in o(gap)$ if the gaps are highly compressible, and prove that $nH_{\text{gap}}^0 \leq n \log(u/n) + n$ bits. This upper bound is smaller than the worst-case size of the Elias $\delta$-encoded gap stream. Our aim is, therefore, to obtain a data structure having $nH_{\text{gap}}^0$ as leading term in its space complexity.

We propose a fully-indexable dictionary that supports rank and select queries in $O(\log \log u)$ time and requires $nH_{\text{gap}}^0 + O(n) + O(u/\text{polylog}(u))$ bits of space. If $n \in \Theta(u/\text{polylog}(u))$ and $H_{\text{gap}}^0 \in O(1)$ (e.g. regularly spaced items), ours is the first solution answering all queries in $O(\log \log n)$ time while requiring only $O(n)$ bits of space.

Keywords: dictionary problem, gap encoding, entropy, compression, rank, select

1 Introduction and Related Work

The dictionary problem on set data asks to maintain a (possibly space-efficient) data structure called indexable dictionary over a set $S = \{s_1, ..., s_n\} \subseteq \{0, ..., u - 1\} = U$, $s_1 < s_2 < ... < s_n$, supporting efficiently a range of queries on $S$. In this problem, $U$ is an ordered set and is called universe. As showed by Jacobson in his doctoral thesis [9], a set of just two operations, rank and select, is sufficient and powerful enough in order to derive other fundamental functionalities desired from such a structure: member, successor, and predecessor. rank($S, x$), with $x \in U$, is the number of elements in $S$ that are small than or equal to $x$. select($S, i$), where $0 \leq i < n$, is the $i$-th smallest element in $S$. In this paper, we focus on fully-indexable dictionaries (FIDs), i.e. data structures supporting both rank and select operations efficiently.

Jacobson in [9] proposed a solution for this problem taking $u + o(u)$ bits of space and supporting constant-time rank. Constant-time select within $o(u)$ bits of additional space was added by Munro [12] and Clark [5]. These results were further improved firstly by Pagh [13] (who considered rank) and then by Raman et al. [15] (rank and select) with structures having the same time complexities and requiring only $\mathcal{B}(n, u) + O(u \log \log u/\log u)$ bits of space, where $\mathcal{B}(n, u) = \lceil \log (\binom{u}{n}) \rceil$ is the minimum number of bits required in order to distinguish any two size-$n$ subsets of $U$. Finally, Pătrașcu [14] reduced the sublinear term to $O(u/\text{polylog}(u))$ while retaining constant query times. Despite these last
results being optimal for big values of $n$, the $o(u)$ term can however be much bigger than $\mathcal{B}(n,u)$ (even exponentially) if $n$ is very small. Moreover, even the $\mathcal{B}(n,u)$ term is not optimal for all instances, and can be improved in many cases of practical interest. To see why this fact holds true, it is sufficient to notice that zero-order entropy compressors encode to the same bit-size all size-$n$ subsets $S$ of $U$, without taking advantage of the structure of $S$ (for example, long or regular distances between its elements). This problem motivates the search for more data-aware measures able to break the $\mathcal{B}(n,u)$ limit in some cases. One of the most widely known such data-aware measures is gap [3], which is defined to be the sum of all bit-lengths of the distances between consecutive elements in $S$. If these distances are not evenly distributed, gap can be much smaller than $\mathcal{B}(n,u)$, reaching 10%-40% of $\mathcal{B}(n,u)$ in some instances of practical interest [8]. By using logarithmic codes such as Elias $\delta$-encoding [6], the stream of gaps can be compressed to $\text{gap} + o(\text{gap})$ bits, where the $o(\text{gap})$ overhead comes from the prefix property of such codes, needed to unambiguously reconstruct codeword boundaries. In [3], Gupta et al. show how to build a FID based on $\delta$-encoding requiring only $\text{gap} + \mathcal{O}(n \log(u/n)/\log n) + \mathcal{O}(n \log \log(u/n))$ bits of space and supporting rank and select in $\mathcal{O}((\log \log u)^2)$—this is nearly optimal within that space, see [12]—and $\mathcal{O}(\log \log n)$ time, respectively. Other recent works [10,16] showed that constant-time queries can be supported using $\text{gap} + \mathcal{O}(n \log \log(u/n)) + o(u)$ bits of space, where the $o(u)$ term is $\mathcal{O}(u \log \log u/\sqrt{\log u})$ in [10] and $\mathcal{O}(u \log \log u/\log u)$ in [16]. These two results are important if $n \in \Omega(u \log \log u/\log u)$, but do not perform well if $n \ll u$ since the $o(u)$ term soon becomes much greater than gap and dominates the overall space of the structure.

$\text{gap}$ reaches its maximum when all gap lengths are equal. However, it is clear that in this scenario other techniques (e.g. zero-order entropy compression) could be flanked to gap encoding in order to turn this worst-case into a $\mathcal{O}(n)$-bits best-case. In this paper we explore the possibility of compressing gaps to their zero-order empirical entropy $H_0^{\text{gap}}$, aiming at obtaining $nH_0^{\text{gap}}$ as leading term in the space complexity of our structures. Similar techniques are already employed in BWT-based text compression algorithms [4], where runs of zeros in the move-to-front encoding of the BWT are compressed using run-length-encoding followed either by zero-order entropy compression or by logarithmic encoding [3] (being runs mostly dominated by small numbers). We firstly observe that $nH_0^{\text{gap}} \in o(\text{gap})$ if gaps are highly compressible, and prove that in any case $nH_0^{\text{gap}}$ does not exceed $n \log(u/n) + n$ bits. This bound coincides with the worst-case of gap, with the difference that (as opposed to gap) $nH_0^{\text{gap}}$ includes information needed to unambiguously reconstruct codeword boundaries. Finally, we simulate several data sets using different gap distributions, and show that in all cases $nH_0^{\text{gap}}$ significantly improves both on gap + $\delta$-encoding and on simple zero-order entropy compression of the set.

These considerations suggest that the data-aware measure $nH_0^{\text{gap}}$ should be preferred to gap in cases where the overhead introduced by the zero-order compressor (e.g. a codebook) is negligible. Our work goes in this direction. We firstly propose a structure that answers all queries in $\mathcal{O}(\log u)$ time while taking
\[ nH_0^{gap} + \mathcal{O}(n) + \mathcal{O}(\log u \log \log u) + \mathcal{O}(\min\{gap, \sqrt{u} \log u\}) \text{ bits of space. Using this solution as building block, we finally propose a FID that answers all queries in } \mathcal{O}(\log \log n) \text{ time and whose space occupancy is of } nH_0^{gap} + \mathcal{O}(n) + \mathcal{O}(u/\text{polylog}(u)) \text{ bits. If } n \in \Theta(u/\text{polylog}(u)) \text{ and } H_0^{gap} \in \Theta(1) \text{ (this can happen, for example, if items are regularly spaced so that there are few distinct gap lengths), ours is the first solution answering all queries in } \mathcal{O}(\log \log n) \text{ time while requiring only } \mathcal{O}(n) \text{ bits of space.} \]

2 Gap-Encoded Dictionaries

In this section we will assume that \( u - 1 \in S \), so that each gap corresponds to an element in \( S \) (i.e. the element following the gap). If \( u - 1 \notin S \), then we can simply use an extra bit to denote this case and encode the final gap length separately. We will moreover assume that \( n \leq u/2 \); otherwise, we can simply invert the roles of 0s and 1s. Logarithms are taken in base 2, unless differently specified. In gap encoding, we represent the set \( S = \{s_1, ..., s_n\} \subseteq \{0, ..., u - 1\} = U, s_1 < s_2 < ... < s_n \) as the stream of gaps \( g_1, ..., g_n \), where \( g_1 = s_1 + 1 \) and \( g_i = s_i - s_{i-1} \) for \( i > 1 \). In order to reduce space occupancy of the stream, variable-length encoding can be used to encode each of the \( g_i \). The data-aware measure \( \text{gap}(S) \) is defined as \( \text{gap}(S) = \sum_{i=1}^{n} (\lceil \log g_i \rceil + 1) \), that is, the total number of bits required in order to store all \( g_i \)'s using the minimum number of bits to represent each gap. When clear from the context, we will simply write \( \text{gap} \) instead of \( \text{gap}(S) \). Clearly, \( S \) cannot be represented using only \( \text{gap} \) bits since we need additional information in order to make the stream uniquely decodable. We adopt a notation similar to [8] and indicate with \( Z_C(S) \)—or simply \( Z_C \) when clear from the context—the decoding overhead (in bits) introduced by the coding scheme \( C \). If we use a separate bitvector \( B \) marking with a 1 the beginning of each code, then we obtain \( Z_B = \text{gap} \). Another solution is to use logarithmic codes such as Elias \( \gamma \) or \( \delta \)-encoding [6]. In \( \gamma \)-encoding, we encode \( \lceil \log g_i \rceil + 1 \) in unary, followed by the \( \lceil \log g_i \rceil \)-bits binary representation of \( g_i \) without the most significant 1. Then, \( Z_\gamma = \text{gap} - n \). A better solution is \( \delta \)-encoding, where we encode with \( \gamma \) the number \( \lceil \log g_i \rceil + 1 \), followed by the \( \lceil \log g_i \rceil \)-bits binary representation of \( g_i \) without the most significant 1. Then, \( Z_\delta = 2 \sum_{i=1}^{n} \lceil \log (\lceil \log g_i \rceil + 1) \rceil \) bits. In this work, we will pay particular attention to the worst-case of the considered data-aware measures. Being log a concave function, the worst-case of \( \text{gap} \) occurs when \( g_1 = g_2 = ... = g_n = u/n \) (by Jensen’s inequality), yielding the upper bounds \( \text{gap} \leq n \log(u/n) + n \) and \( Z_\delta \leq 2n \log \log(u/n) \). Then, one can prove the following (for the original proof, see [7]):

**Lemma 1.** \( \text{gap} \leq B(n, u) \) if \( n \leq u/2 \).

**Proof.** The claim follows directly from \( \text{gap} \leq n \log(u/n) + n \) and from the fact that \( B(n, u) = n \log(u/n) + n \log e - \Theta(n/u) + \mathcal{O}(\log n) \) if \( n \leq u/2 \).

Moreover, let \( H_0 = \frac{n}{u} \log \left( \frac{n}{u} \right) + \frac{u - n}{u} \log \frac{u}{u - n} \) be the zero-order empirical entropy of the set \( S \). Since \( B(n, u) \leq uH_0 \), we have that:

**Corollary 1.** \( \text{gap} \leq uH_0 \) if \( n \leq u/2 \).
The above inequalities are extremely important as they show that gap encoding can never perform worst than zero-order entropy compression. On the other hand, experiments show [8] that gap can be significantly smaller than $B(n, u)$ for many cases of interest, thus motivating its use in practical applications. In the following section we take one step forward, exploring what happens when we treat $S$ as a sequence on the alphabet $\{g_1, ..., g_n\}$ and then apply zero-order entropy compression to it.

### 2.1 A Compressed-Gap Data-Aware Measure

$gap$ reaches its worst-case of $n \log(u/n) + n$ bits when all gaps have the same length. However, it is clear that entropy compression should turn this worst-case scenario into a best-case, being the zero-order empirical entropy of such a configuration equal to 0. More formally, let’s consider the following representation $T_S$ of $S$. We define $T_S$ to be the sequence $g_1g_2...g_n \in \Sigma^n$, where $\Sigma_{\text{gap}} = \{g_1, g_2, ..., g_n\}$. Let moreover $\sigma = |\Sigma_{\text{gap}}|$ be the alphabet size and $f(s) = \text{occ}(s)/n$, $s \in \Sigma_{\text{gap}}$, be the empirical frequency of $s$ in $T_S$, where $\text{occ}(s)$ is the number of occurrences of $s$ in $T_S$. We define the zero-order empirical entropy of the gaps $H_{gap}^0$ to be

**Definition 1.** $H_{gap}^0 = - \sum_{s \in \Sigma_{\text{gap}}} f(s) \log(f(s))$

$nH_{gap}^0$ is the minimum number of bits output by any compressor that encodes $T_S$ assigning a unique code to each symbol in $\Sigma_{\text{gap}}$. First of all, we observe that $nH_{gap}^0$ can be significantly smaller than $gap$: if $g_1 = g_2 = ... = g_n = u/n$, then $n \log(u/n) \leq gap \leq n \log(u/n) + n$ and $nH_{gap}^0 = 0$. Moreover, $nH_{gap}^0$ is never worst than the length of any decodable gap-compressed sequence:

**Lemma 2.** $nH_{gap}^0 \leq gap + Z_C$, where $C$ is any prefix coding scheme.

**Proof.** Follows directly from the fact that no prefix code can compress $T_S$ in less than $nH_{gap}^0$ bits. \[\square\]

Using Lemma 2 and the bounds for $gap$ and $Z_\delta$ derived in the previous section, one can obtain $H_{gap}^0 \leq \log(u/n) + 2 \log \log(u/n) + 1$. With the following theorem we show a much stronger upper bound:

**Theorem 1.** $H_{gap}^0 \leq \log(u/n) + 1$

**Proof.** We want to compute

$$\max_{\Sigma_{\text{gap}} \subseteq \mathbb{N} > 0} \max_{f : \Sigma_{\text{gap}} \to \mathbb{R}^+} H_{gap}^0$$

where the alphabet $\Sigma_{\text{gap}}$ and the empirical frequency function $f$ must satisfy:

$$n \sum_{s \in \Sigma_{\text{gap}}} f(s) \cdot s = u \quad (1)$$

4
Let $\sigma = |\Sigma_{\text{gap}}|$. From Definition 1 and from the concavity of log, we have that $H_{0}^{\text{gap}}$ reaches its maximum $H_{0}^{\text{gap}} = \log \sigma$ when all frequencies are equal, i.e. $f(s) = \sigma^{-1}$ for all $s \in \Sigma_{\text{gap}}$. We thus have

$$\max_{\Sigma_{\text{gap}} \subseteq \mathbb{N}_{>0}} \max_{f: \Sigma_{\text{gap}} \to \mathbb{R}^{+}} H_{0}^{\text{gap}} = \max_{\Sigma_{\text{gap}} \subseteq \mathbb{N}_{>0}, f(s)=\sigma^{-1}} \log \sigma$$

In order to maximize $\log \sigma$, we now have to find $\Sigma_{\text{gap}}$ of maximum cardinality that satisfies condition (1). It is easy to see that $\Sigma_{\text{gap}} = \{1, ..., \sigma\}$ minimizes $\sum_{s \in \Sigma_{\text{gap}}} s = \sum_{i=1}^{\sigma} i = \sigma(\sigma + 1)/2$. Since, moreover, $f(s) = \sigma^{-1}$ for all $s \in \Sigma_{\text{gap}}$, we can rewrite (1) as $n\sigma^{-1}(\sigma(\sigma + 1)/2 + k) = u$, where $k \geq 0$. Solving in $\sigma$, we obtain the set of solutions

$$Z = \left\{ \left( b \pm \sqrt{b^2 - 8kn^2} / (2n) \right) \middle| b = 2u - n \land k \geq 0 \right\}$$

for which we have $\max Z = (2u - n)/n$ in $k = 0$. This implies that $\Sigma_{\text{gap}} = \{1, ..., (2u - n)/n\}$ and $f(s) = n/(2u - n)$ for all $s \in \Sigma_{\text{gap}}$ maximize $H_{0}^{\text{gap}}$. Our claim follows:

$$H_{0}^{\text{gap}} \leq \log \frac{2u - n}{n} = \log(u/n) + \log \frac{2u - n}{u} \leq \log(u/n) + 1$$

Interestingly, the two measures gap and $nH_{0}^{\text{gap}}$ are upper-bounded by the same quantity $n \log(u/n) + n$. This is not a trivial result since, differently from $nH_{0}^{\text{gap}}$, gap does not include information needed to reconstruct unambiguously codeword boundaries (even though $nH_{0}^{\text{gap}}$, in turn, does not include information—e.g. a codebook—needed to decode codewords). Using the same arguments of Lemma 1 and Corollary 1, we can moreover derive the bounds:

**Corollary 2.** $nH_{0}^{\text{gap}} \leq B(n, u) \leq u H_{0}$ if $n \leq u/2$

In order to assess also in practice the differences between the above discussed measures, we adopted the approach of [7] and simulated several sets, computing for each of them the measures gap, gap + $Z_{\delta}$, $uH_{0}$, and $nH_{0}^{\text{gap}}$. Gaps were generated according to uniform (Table 1) and binomial (Table 2) distributions. Table 1 reports the same experiment performed in [7] (except from the facts that we use $\delta$ instead of $\gamma$ and we do not consider RLE), updated with our measure $nH_{0}^{\text{gap}}$. Despite this being its worst-case (uniform gaps), $nH_{0}^{\text{gap}}$ is always very close to gap, and considerably improves upon $\delta$-encoding. The advantages of $nH_{0}^{\text{gap}}$, however, are more evident if non-uniform distributions are used. Table 2 reports the results on binomially-distributed gap. As expected, in this case our measure considerably improves even on gap, reaching 63% of it in some cases. When compared with $\delta$-encoding, this fraction drops down to 38%.

---

1. We chose a binomial distribution in order to model a scenario in which gap lengths are accumulated around a value $\mu \gg 0$ (in this case, $\mu$ is the mean). Intuitively, in this case gap does not perform well because small numbers are not frequent.
\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|}
\hline
\text{log(max_gap)} & \text{gap} & \text{gap + } Z_\delta & uH_0 & nH^{\text{gap}}_0 \\
\hline
1 & 1.6667 & 3.0001 & 2.00016 & 1.58496 \\
2 & 2.19732 & 3.79436 & 2.75233 & 2.3219 \\
3 & 2.78069 & 5.00409 & 3.61219 & 3.16987 \\
4 & 3.4715 & 6.53214 & 4.53051 & 4.08737 \\
5 & 4.27697 & 7.79269 & 5.48949 & 5.04416 \\
6 & 5.15021 & 8.90383 & 6.46356 & 6.02165 \\
7 & 6.08147 & 9.99053 & 7.45098 & 7.01024 \\
8 & 7.04877 & 11.9999 & 8.44942 & 8.0039 \\
9 & 8.01987 & 13.4907 & 9.44336 & 9.99876 \\
10 & 9.02043 & 14.7615 & 10.4451 & 10.99419 \\
11 & 10.004 & 15.8709 & 11.4419 & 10.9868 \\
12 & 11.077 & 16.9431 & 12.4467 & 11.9712 \\
13 & 12.035 & 17.9705 & 13.4427 & 12.9413 \\
14 & 13.016 & 18.9853 & 14.4401 & 13.8781 \\
15 & 14.054 & 19.9973 & 15.4442 & 14.74 \\
\hline
\end{tabular}
\caption{Comparison between \text{gap}, \text{gap + } Z_\delta, \text{uH}_0, \text{and } nH^{\text{gap}}_0 \text{ on randomly-generated sets. Gaps between the } n \text{ items (} n \text{ affects only the variance of the results; we used } n = 10^5 \text{) are uniformly distributed in the interval } [1, \text{max_gap}]. \text{ All columns except the first report the number of bits per item required by each method.}}
\end{table}

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|}
\hline
\text{log(max_gap)} & \text{gap} & \text{gap + } Z_\delta & uH_0 & nH^{\text{gap}}_0 \\
\hline
1 & 1.74887 & 3.24661 & 1.99692 & 1.49916 \\
2 & 2.24864 & 4.12432 & 2.75332 & 2.02899 \\
3 & 2.88324 & 4.94294 & 3.60709 & 2.5452 \\
4 & 3.76998 & 7.31386 & 4.52697 & 3.9416 \\
5 & 4.70257 & 8.70203 & 5.48985 & 3.5447 \\
6 & 5.64642 & 9.64642 & 6.4639 & 4.04007 \\
7 & 6.60588 & 10.6059 & 7.45466 & 4.54554 \\
8 & 7.5753 & 12.7259 & 8.44846 & 5.04485 \\
9 & 8.55103 & 14.5511 & 9.4452 & 5.54811 \\
10 & 9.53602 & 15.536 & 10.444 & 6.04794 \\
11 & 10.528 & 16.528 & 11.4435 & 6.5462 \\
12 & 11.5148 & 17.5148 & 12.4429 & 7.04787 \\
13 & 12.5115 & 18.5115 & 13.4428 & 7.54833 \\
14 & 13.5096 & 19.5096 & 14.4428 & 8.04141 \\
15 & 14.5068 & 20.5068 & 15.4427 & 8.5447 \\
\hline
\end{tabular}
\caption{Comparison between \text{gap}, \text{gap + } Z_\delta, \text{uH}_0, \text{and } nH^{\text{gap}}_0 \text{ on randomly-generated sets. Gaps between the } n \text{ items (} n = 10^5 \text{) are binomially distributed in the (shifted) interval } [1, \text{max_gap}]. \text{ All columns except the first report the number of bits per item required by each method.}}
\end{table}
3 A Compressed-Gap FID

Let us now turn our attention to fully-indexable dictionary data structures. Our aim is to obtain a structure whose space occupancy is of the form $nH_0^{gap} + O(n)$ bits. In order to improve space requirements of other gap-based solutions, we want to avoid second-order terms of the form $\Omega(n)$ (which can be bigger than $nH_0^{gap}$). We show a solution that reaches this objective for dense enough sets.

Our strategy is the following: we still use Elias $\delta$-encoding, but we exploit its property of being an asymptotically optimal universal code \[6\] to encode the gap stream in $nH_0^{gap} + O(n)$ bits. We then build a two-levels structure atop of this representation to support rank and select queries. We adopt an approach similar to \[8\] and firstly describe a binary-searchable dictionary (BSD) that supports all queries in $O(\log u)$ time. The BSD is finally used as building block for our final structure, which improves all query times to $O(\log \log u)$.

Let $\Sigma_{gap}$ and $f : \Sigma_{gap} \rightarrow \mathbb{R}^+$ be the set of all gap lengths and the empirical frequencies associated with the gap stream, respectively, and consider an (arbitrary) ordering of the symbols $\text{ord} : \Sigma_{gap} \rightarrow \{1, ..., \sigma\}$, $\sigma = |\Sigma_{gap}|$ (i.e. a bijection) such that if $\text{ord}(g_i) < \text{ord}(g_j)$ then $f(g_i) \leq f(g_j)$ for all $g_i, g_j \in \Sigma_{gap}$. Let $\delta(x)$, $x > 0$ be the Elias $\delta$ code of the integer $x$. Then, we associate the code $\text{code}(g_i) = \delta(\text{ord}(g_i))$ to each gap length $g_i \in \Sigma_{gap}$. Being $\delta$ an asymptotically optimal universal code \[6\], the bit length $l$ of the compressed stream $\text{code}(g_1) \ldots \text{code}(g_n)$ is equal to $nH_0^{gap} + O(n)$ bits. In the following we assume to work under the word RAM model with word size $\Theta(\log u)$ bits, so that we can extract any $O(\log u)$-bits block from a plain bitvector in constant time. We store the bit representations of the compressed gaps sequentially in a bitvector $C[0, ..., l - 1] = \text{code}(g_1) \ldots \text{code}(g_n)$. An additional array $D[1, ..., \sigma]$ defined as $D[i] = \text{ord}^{-1}(i)$ (the codebook) is moreover built to permit the decoding of codewords. Note that $\sigma \in O(\sqrt{u})$ \[8\], so $D$ takes $O(\sqrt{n}\log u)$ bits of space. To further reduce its space occupancy, $D$ can be compressed using gap encoding and augmented with a constant-time rank and select succinct bitvector marking code boundaries to permit $O(1)$ access to $D$’s entries. In the worst case, each gap length occurs only once in $T_S$; the total bit size of $D$ is thus upper bounded by $O(\min\{\sqrt{n}\log u, gap\})$. That is, given the starting position of $\text{code}(g_i)$, $0 \leq i < n$, in the bitvector $C$, we can extract and decode $\text{code}(g_i) = \delta(\text{ord}(g_i))$ in $O(1)$ time: firstly, we need to decode the $\gamma$-prefix of $\delta(\text{ord}(g_i))$. This can be done in $O(1)$ time using two universal tables of $O(2^{\log \log u}\log \log u) = O(\log u \log \log u)$ bits each (one for the unary prefix and the other for the rest of the $\gamma$-prefix of the code). This gives us (i) the bit-length of the $\gamma$-prefix of $\delta(\text{ord}(g_i))$, and (ii) the bit-length of $\text{ord}(g_i)$ (without the most significant bit). We can then extract the bits of $\text{ord}(g_i)$ and access $D[\text{ord}(g_i)] = g_i$ in constant time. To improve readability, in the next sections we will implicitly make use of this strategy and—provided that we know the starting position of $\text{code}(g_j)$ in $C$—say read gap $g_j$ instead of extract and decode $\text{code}(g_j)$.

\[8\] assume, by contradiction, that $\sigma \in \Omega(\sqrt{u})$. Then, the alphabet that minimizes $\sum_{s \in \Sigma_{gap}} s$ is $\Sigma_{gap} = \{1, ..., \sigma\}$, for which we obtain $\sum_{s \in \Sigma_{gap}} s = \Theta(\sigma^2) = \Omega(u)$. This is an absurd since the sum of all gaps cannot exceed $u$. 

7
3.1 A Binary-Searchable Dictionary

We divide the elements of $S = \{s_1, ..., s_n\}$ in blocks of size $t = \lceil \log u \rceil$ (we assume for clarity of exposition that $t$ divides $n$; the following arguments can be easily adapted to the general case). For each block $\{s_{it+1}, ..., s_{(i+1)t}\}$, $i = 0, ..., n/t - 1$, we store explicitly the smallest element $s_{it+1}$ ($O(\log u)$ bits) and a pointer to the beginning of $\text{code}(g_{it+2})$ in the bitvector $C[3] (O(\log u)$ bits). These structures are sufficient to obtain our BSD. $\text{select}(S, i), 0 \leq i < n$, is implemented by accessing the $[i/t]$-th block and reading $i \bmod t < t$ gaps in $C$ starting from $g_{[i/t]t+2}$.

Then,

$$\text{select}(S, i) = s_{[i/t]t+1} + \sum_{j=[i/t]t+2}^{i+1} g_j$$

$\text{rank}(S, x), x \in U = \{0, ..., u - 1\}$, is implemented by binary-searching the blocks according to explicitly stored elements $s_{it+1}, i = 0, ..., n/t - 1$, and then by extracting gaps in the block of interest until we reach element $x$. More formally, let $0 \leq i \leq n/t - 1$ be the biggest integer (if any) such that $s_{it+1} \leq x$. $i$ can be found by binary search in $O(\log u)$ time. If such an integer does not exist, then $\text{rank}(S, x) = 0$. Otherwise, let $1 \leq j < t$ be the smallest integer such that $q = s_{it+1} + \sum_{h=1}^{j} g_{it+1+h} \geq x$. $j$ can be found by linear search in $O(t) = O(\log u)$ time. Then,

$$\text{rank}(S, x) = \begin{cases} it + j + 1 & \text{if } q = x \\ it + j & \text{if } q > x \end{cases}$$

Since for each block we spend $O(\log u)$ bits (one element $s_{it}$ and a pointer to $C$), the blocks take overall $O(\log u \cdot n / \log u) = O(n)$ bits. The overall size of our BSD structure is thus $nH^\text{gap}_0 + O(n) + O(\log u \log \log u) + O(\min\{\sqrt{u} \log u, \text{gap}\})$ bits, where the last two terms come from the universal table (needed to decode $\delta$) and from the codebook. We state this result in the following lemma:

**Lemma 3.** The binary-searchable dictionary described in section 3.1 occupies $nH^\text{gap}_0 + O(n) + O(\log u \log \log u) + O(\min\{\sqrt{u} \log u, \text{gap}\})$ bits of space and supports rank and select queries in $O(\log u)$ time.

Note that the size of the proposed BSD can be exponentially smaller than $u$. However, query times are not particularly interesting and can be improved. In the next section we show how to obtain $O(\log \log u)$-time queries while incurring in a $o(u)$ space overhead.

3.2 A Fully-Indexable Dictionary

Let $k \geq 1$ be a constant and $v = \lceil \log^{k+1} u \rceil$. The idea is to divide $U$ in blocks of $v$ elements, and store a BSD for each block.

We build a constant-time rank and select succinct bitvector $V[0, ..., \lceil u / v \rceil - 1]$ defined as $V[i] = 1$ if and only if $S \cap \{iv, ..., (i+1)v - 1\} \neq \emptyset$. Additionally, one array $R[0, ..., \lceil u / v \rceil - 1]$ stores sampled rank results: $R[0] = 0$ and $R[i] = \text{code}(g_{it+2})$ instead of $\text{code}(g_{it+1})$ because $s_{it+1}$ is explicitly stored. As a matter of fact, we can avoid storing $\text{code}(g_{it+1})$ in $C$. 

---

\(^3\) We point to $\text{code}(g_{it+2})$ instead of $\text{code}(g_{it+1})$ because $s_{it+1}$ is explicitly stored. As a matter of fact, we can avoid storing $\text{code}(g_{it+1})$ in $C$. 

rank(S, iv − 1) for i > 0. We build a binary-searchable dictionary BSD(i) for each set \(S_i = \{x - iv \mid x \in S \cap \{iv, ..., (i+1)v - 1\}\}, \ i = 0, ..., \lfloor u/v \rfloor - 1\), where we use the same codebook \(D\) for all the BSD structures (i.e. \(D\) is computed according to all gaps \(g_1, ..., g_n\)). Note that there may exist a set \(S_i\) (or more than one) such that its first gap does not belong to \(\{g_1, ..., g_n\}\). This happens each time an element \(s_i\) is the first of its block \(b = \lfloor s_i/v \rfloor > 0\), the gap \(g_i\) overlaps blocks \(b \) and \(b - 1\), and \(s_i - b \cdot v + 1 \notin \{g_1, ..., g_n\}\). However, by construction of the BSD data structure (see previous section), the first gap in \(S_i\) is never used (since we store the smallest element of \(S_i\) explicitly), so this event does not affect overall gap frequencies nor space requirements of the array \(D\). Finally, one array \(SEL[0, ..., \lceil n/t \rceil - 1]\), where \(t = \lceil \log u \rceil\), stores the (number of the) block containing \(s_{it+1}\): \(SEL[i] = \lfloor s_{it+1}/v \rfloor\), for \(i = 0, ..., \lceil n/t \rceil - 1\).

Using the above described structures, we can now show how to efficiently solve queries. \(\text{rank}(S, x), \ x \in U = \{0, ..., u - 1\}\), is implemented by accessing the \([x/v]-\text{th}\) block and calling \text{rank} on BSD([x/v]). More formally,

\[
\text{rank}(S, x) = R([x/v]) + \text{rank}(S_{[x/v]}, x \mod v)
\]

where \(\text{rank}(S_{[x/v]}, x \mod v)\) is called on the structure BSD([x/v]). Rank is thus solved in \(O(\log v) = O(k \log \log u)\) time. To solve \(\text{select}(S, i)\), we firstly find by binary search the block containing \(s_{i+1}\), and then call \text{select} on the corresponding BSD. More in detail, let \(q_l = SEL[\lceil i/t \rceil]\) and \(q_r = SEL[\lceil i/t \rceil + 1]\) if \([i/t] + 1 < \lceil n/t \rceil\), \(q_r = q_l\) otherwise. By construction of \(SEL\), the block containing element \(s_{i+1}\) is one of \(q_l, q_l + 1, ..., q_r\). Note that the number \(q_r - q_l + 1\) of blocks of interest can be arbitrary large since there may be an arbitrary number of empty blocks among them. However, at most \(t\) of them will contain at least one element (by construction of \(SEL\)). Then, we can perform binary search only on the blocks marked with a 1 in the array \(V\): during binary search we access blocks at positions of the form \(select(V, j)\) (note: this is a constant-time select performed on the bitvector \(V\)), starting with the range \(j \in [\text{rank}(V, q_l) - 1, \text{rank}(V, q_r) - 1]\). Binary search is performed according to partial ranks (array \(R\)). Let \(q_l \leq q_m \leq q_r\) be the biggest integer such that \(R[q_m] \leq i < R[q_m + 1]\) (if \(q_m + 1 \geq \lceil u/v \rceil\) then simply ignore the upper bound in the previous inequality). According to the above considerations, \(q_m\) can be found in \(O(\log t) = O(\log \log u)\) time using binary search. We can solve \(\text{select}(S, i)\) as follows:

\[
\text{select}(S, i) = q_m \cdot v + \text{select}(S_{qm}, i - R[q_m])
\]

where \(\text{select}(S_{qm}, i - R[q_m])\) is called on the structure BSD(qm). select is thus solved on our FID in \(O(\log \log u) + O(\log v) = O(k \log \log u)\) time (since \(k \geq 1\)).

Bitvector \(V\) can be implemented using \((1 + o(1))u/v = (1 + o(1))u/\log^{k+1} u\) bits. Arrays \(R\) and \(SEL\) take \(\log u \cdot u/v = u/\log^k u\) and \(\log u \cdot n/t = n\) bits of space, respectively. Finally, all BSD data structures take overall \(nH_0^{gap} + O(n)\) bits, and the codebook \(D\) and the universal tables take \(O(\min\{\sqrt{u \log u, gap}\}) \leq O(u/\log^k u)\) and \(O(\log u \log \log u) \leq O(u/\log^k u)\) bits, respectively. We can state our final result:
Theorem 2. The structure described in section 3.2 is a fully-indexable dictionary occupying $nH_0^{gap} + O(n) + O(u/\log^k u)$ bits of space while supporting rank and select queries in $O(k \log \log u)$ time, for any constant $k \geq 1$.

The result stated in Theorem 2 improves the space of \[10,16\], reducing both leading and $o(u)$ terms from $gap + O(n \log \log (u/n))$ and $u \log \log u/\log u$ bits to $nH_0^{gap} + O(n)$ and $O(u/polylog(u))$ bits, respectively. This spatial improvement comes at the price of a $O(\log \log u)$ slowdown in all query times. Notice that we cannot apply the general technique proposed by Mäkinen and Navarro in \[10\] in order to obtain $O(1)$ query times since $\text{code}(\cdot)$ does not (always) satisfy $|\text{code}(x)| \in O(\log x)$ (this is one of the properties characterizing random access self-delimiting codes \[10\]). An interesting line of research would be to envision a broader class of codes (including $\text{code}(\cdot)$) for which we can describe a general technique guaranteeing constant-time queries.

4 Conclusions

In this paper we explored the possibility of improving space requirements of fully-indexable dictionaries by using a compressed-gap data-aware measure—$nH_0^{gap}$—that combines gap encoding and zero-order entropy compression. We provided new theoretical upper-bounds for this measure, and showed that in practice $H_0^{gap}$ always improves space usage of gap encoding techniques combined with logarithmic codes such as Elias $\delta$-encoding. Finally, we proposed a compressed-gap fully-indexable dictionary supporting fast queries ($O(\log \log u)$-time rank and select) and taking small space in addition to $nH_0^{gap}$ ($O(n) + O(u/polylog(u))$ bits of redundancy).

As expected, simulations confirmed that the proposed compressed-gap measure is particularly convenient in situations where the gaps follow a non-uniform distribution or they are dominated mainly by large numbers, and improves on gap+$\delta$ even in the case of uniform gaps. The main drawback of $nH_0^{gap}$ seems to be the overhead introduced by the zero-order compressor, which in our solution is of $\Theta(\sqrt{u} \log u)$ bits in the worst case. However, in some practical applications this overhead—being proportional to the number of distinct gap lengths—is expected to be negligible with respect to the overall structure size. One example of such an application is run-length compression of the BWT of highly repetitive text collections (e.g. genome variants), where run lengths are expected to scale linearly with the number of documents in the collection \[11,17\]. Another concern is the $o(u)$ term in the space usage, which can be considerably larger than $nH_0^{gap}$ for sparse instances. This is a problem shared with other solutions (e.g. \[10,16\]), and cannot be completely solved while at the same time maintaining the same query time bounds \[12\].

Future work directions include an implementation and practical evaluation of the proposed full-indexable dictionary and theoretical developments such as (i) improving the $o(u)$ term while still supporting efficient queries, (ii) obtaining constant-time queries within small redundancy space (e.g. by using a general scheme similar to the one proposed in \[10\]), and (iii) reducing the impact of the codebook size (e.g. by making assumptions on the gap probability distribution).
References

1. Andersson, A.A., Thorup, M.: Tight (er) worst-case bounds on dynamic searching and priority queues. In: Proceedings of the thirty-second annual ACM symposium on Theory of computing. pp. 335–342. ACM (2000)
2. Beame, P., Fich, F.E.: Optimal bounds for the predecessor problem. In: Proceedings of the thirty-first annual ACM symposium on Theory of computing. pp. 295–304. ACM (1999)
3. Bell, T.C., Moffat, A., Nevill-Manning, C.G., Witten, I.H., Zobel, J.: Data compression in full-text retrieval systems. Journal of the American Society for Information Science 44(9), 508–531 (1993)
4. Burrows, M., Wheeler, D.J.: A block-sorting lossless data compression algorithm. Tech. rep., Digital Equipment Corporation (1994)
5. Clark, D.: Compact Pat trees. Ph.D. thesis, University of Waterloo (1996)
6. Elias, P.: Universal codeword sets and representations of the integers. Information Theory, IEEE Transactions on 21(2), 194–203 (1975)
7. Grossi, R., Gupta, A., Vitter, J.S.: When indexing equals compression: Experiments with compressing suffix arrays and applications. In: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms. pp. 636–645. Society for Industrial and Applied Mathematics (2004)
8. Gupta, A., Hon, W.K., Shah, R., Vitter, J.S.: Compressed data structures: Dictionaries and data-aware measures. In: Data Compression Conference, 2006. DCC 2006. Proceedings. pp. 213–222. IEEE (2006)
9. Jacobson, G.J.: Succinct static data structures. Ph.D. thesis, Dept. of Computer Science, Carnegie Mellon University (1988)
10. Mäkinen, V., Navarro, G.: Rank and select revisited and extended. Theoretical Computer Science 387(3), 332–347 (2007)
11. Mäkinen, V., Navarro, G., Sirén, J., Välimäki, N.: Storage and retrieval of highly repetitive sequence collections. Journal of Computational Biology 17(3), 281–308 (2010)
12. Munro, J.I.: Tables. In: Foundations of Software Technology and Theoretical Computer Science. pp. 37–42. Springer (1996)
13. Pagh, R.: Low redundancy in static dictionaries with constant query time. SIAM Journal on Computing 31(2), 353–363 (2001)
14. Pătraşcu, M.: Succincter. In: Foundations of Computer Science, 2008. FOCS’08. IEEE 49th Annual IEEE Symposium on. pp. 305–313. IEEE (2008)
15. Raman, R., Raman, V., Rao, S.S.: Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms. pp. 233–242. Society for Industrial and Applied Mathematics (2002)
16. Sadakane, K., Grossi, R.: Squeezing succinct data structures into entropy bounds. In: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm. pp. 1230–1239. ACM (2006)
17. Sirén, J.: Compressed full-text indexes for highly repetitive collections. Ph.D. thesis, University of Helsinki, Department of Computer Science (2012)