On representing the degree sequences of sublogarithmic-degree Wheeler graphs

Travis Gagie

Abstract. We show how to store a searchable partial-sums data structure with constant query time for a static sequence S of n positive integers in $o\left(\frac{\log n}{(\log\log n)^2}\right)$, in $nH_k(S) + o(n)$ bits for $k \in o\left(\frac{\log n}{(\log\log n)^2}\right)$. It follows that if a Wheeler graph on n vertices has maximum degree in $o\left(\frac{\log n}{(\log\log n)^2}\right)$, then we can store its in- and out-degree sequences D_{in} and D_{out} in $nH_k(D_{\mathsf{in}})+o(n)$ and $nH_k(D_{\mathsf{out}})+o(n)$ bits, for $k \in o\left(\frac{\log n}{(\log\log n)^2}\right)$, such that querying them for pattern matching in the graph takes constant time.

1 Introduction

A Wheeler graph [7] is a directed edge-labelled graph whose vertices can be ordered such that vertices with no in-edges come first; if u has an in-edge labelled a and v has an in-edge labelled b with $a \prec b$ then $u < v$; if edges (u, v) and (w, x) are both labelled a and $u < w$ then $v \leq x$. Wheeler graphs are interesting because graphs that arise in some important applications are Wheeler — such as collections of edge-labelled paths and cycles, tries, and de Bruijn graphs — and if a graph is Wheeler then we can build a small index for it such that, given a pattern, we can quickly tell which vertices can be reached by paths labelled with that pattern.

The index for a Wheeler graph consists of four components:

1. a data structure supporting sum queries on the list D_{out} of the vertices' out-degrees, with $D_{\mathsf{out}}.\mathsf{sum}(i)$ returning the ith partial sum of the out-degrees (that is, the sum of the out-degrees of the first i vertices in the Wheeler order);
2. a data structure supporting rank queries on the list L of edge labels sorted by the edges' origins, with $L.\mathsf{rank}_a(i)$ returning the frequency of a among the first i edge labels;
3. a data structure supporting sum queries on the list C of the edge labels' frequencies, with $C.\mathsf{sum}(a)$ returning the sum of the frequencies of the edge labels lexicographically strictly less than a;
4. a data structure supporting search queries on the list D_{in} of the vertices' in-degrees, with $D_{\mathsf{in}}.\mathsf{search}(j)$ returning the largest i such that the sum of the in-degrees of the first i vertices in the Wheeler order is at most j.

To see how the index works, first notice that, by the definition of a Wheeler graph, the vertices reachable by paths labelled with a pattern P form a single interval in the Wheeler order. In particular, all the vertices are reachable by paths labelled with the empty string. Suppose we have already found the endpoints $V_P.\mathsf{start}$ and $V_P.\mathsf{end}$ of the interval V_P in the Wheeler order containing vertices reachable by paths labelled P, and we want to find the endpoints $V_{P \cdot a}.\mathsf{start}$ and $V_{P \cdot a}.\mathsf{end}$ of the interval $V_{P \cdot a}$ in the Wheeler order containing vertices reachable by paths labelled $P \cdot a$, where $\cdot$ denotes concatenation.

We use the first data structure to find the endpoints $E_{P,\,\mathrm{out}}.\mathsf{start} = D_{\mathsf{out}}.\mathsf{sum}(V_{P \cdot a}.\mathsf{start}) + 1$ and $E_{P,\,\mathrm{out}}.\mathsf{end} = D_{\mathsf{out}}.\mathsf{sum}(V_P.\mathsf{end})$ of the interval $E_{P,\,\mathrm{out}}$ in L that contains the labels of the out-edges of the vertices in V_P. We then use the second and third data structures to find the endpoints $E_{P \cdot a}.\mathsf{start} = L.\mathsf{rank}_a(E_{P,\,\mathrm{out}}.\mathsf{start} - 1) + 1 + C.\mathsf{sum}(a)$ and $E_{P \cdot a}.\mathsf{end} = L.\mathsf{rank}_a(E_{P,\,\mathrm{out}}.\mathsf{end}) + C.\mathsf{sum}(a)$ of the interval $E_{P \cdot a}$ in the list of edge labels sorted into lexicographic order with ties broken by origin, that contains the copies of a in $E_{P,\,\mathrm{out}}$. Finally, we use the fourth data structure to find the endpoints $V_{P \cdot a}.\mathsf{start} = D_{\mathsf{in}}.\mathsf{search}(E_{P \cdot a}.\mathsf{start})$ and $V_{P \cdot a}.\mathsf{end} = D_{\mathsf{in}}.\mathsf{search}(E_{P \cdot a}.\mathsf{end})$ of $V_{P \cdot a}$.

This works because, again by the definition of a Wheeler graph, the list of edge labels sorted into lexicographic order with ties broken by origin, is also sorted by the ranks in the Wheeler order of the edges' destinations. For the sake of brevity, however, we refer the reader to Gagie et al.'s [7] original paper on Wheeler graphs for a full proof of correctness, and offer here only the example in Figure 1 (modified from [4]), which shows the BOSS [5] representation of a de Bruijn graph (with the out-edge leaving vertex ACT and labelled \$ deleted).

Suppose we have already found the endpoints $V_{\mathsf{C}}.\mathsf{start} = 4$ and $V_{\mathsf{C}}.\mathsf{end} = 6$ of the interval V_{C} of vertices reachable by paths labelled C, and we want to find the endpoints $V_{\mathsf{CG}}.\mathsf{start} = 7$ and $V_{\mathsf{CG}}.\mathsf{end} = 8$ of the interval V_{CG} of vertices reachable by paths labelled CG. We compute

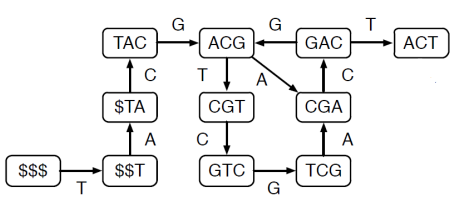
$$\begin{aligned} D_{\mathsf{out}}.\mathsf{sum}(4-1) + 1 &= 4 \\ D_{\mathsf{out}}.\mathsf{sum}(6) &= 7 \end{aligned}$$

$$\begin{aligned} L.\mathsf{rank}_{\mathsf{G}}(4-1) + 1 + C.\mathsf{sum}(\mathsf{G}) &= 7 \\ L.\mathsf{rank}_{\mathsf{G}}(7) + C.\mathsf{sum}(\mathsf{G}) &= 9 \end{aligned}$$

$$\begin{aligned} D_{\mathsf{in}}.\mathsf{search}(7) &= 7 \\ D_{\mathsf{in}}.\mathsf{search}(9) &= 8 \end{aligned}$$

and correctly conclude $V_{\mathsf{CG}} = [7, 8]$ (containing vertices ACG and TCG).

There have been many papers on how to represent L compactly while supporting fast rank queries on it, and representing C compactly while supporting fast sum queries on it is trivial unless the alphabet of edge labels is unusually large, so in this paper we focus on how to represent D_{out} and D_{in} compactly while supporting fast sum and search queries on them. Specifically, we describe the first searchable partial-sums data structure for a static sequence S of sublog-



Wheeler

rank	vertex	D_{out}	L	D_{in}
1)	$\$\$\$$	1	T	0
2)	CGA	1	C	2
3)	$\$TA$	1	C	1
4)	GAC	2	G T	1
5)	TAC	1	G	1
6)	GTC	1	G	1
7)	ACG	2	G T	2
8)	TCG	1	A	1
9)	$\$\$T$	1	A	1
10)	ACT	0		1
11)	CGT	1	C	1

a	$C.\mathsf{sum}(a)$
A	0
C	3
G	6
T	9

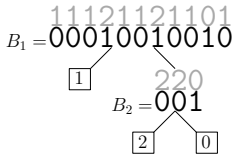


Fig. 1. A Wheeler graph **(upper left; [4])**; a table with D_{out}, L and D_{in} **(right)**; C **(lower left)**; and a degenerate wavelet tree supporting sum on D_{out} **(lower center)**.

arithmic positive integers, with constant query time and space bounded in terms of the kth-order empirical entropy $H_k(S)$ of S:

Theorem 1. *Let $S[1..n]$ be a static sequence of positive integers. If $\max_i\{S[i]\}, k \in o\left(\frac{\log n}{(\log\log n)^2}\right)$ then we can store S in $nH_k(S) + o(n)$ bits and support* sum *and* search *queries on it in constant time.*

Theorem 1 may be of independent interest and it is easy to apply to support sum queries on D_{out} and search queries on D_{in}. To see how we can apply it to D_{out}, notice that if D'_{out} is the sequence obtained from D_{out} by incrementing each out-degree, then D'_{out} contains only positive integers, $|D'_{\mathsf{out}}|H_k(D'_{\mathsf{out}}) = |D_{\mathsf{out}}|H_k(D_{\mathsf{out}})$ and $D_{\mathsf{out}}.\mathsf{sum}(i) = D'_{\mathsf{out}}.\mathsf{sum}(i) - i$. To see how we can apply it to D_{in}, notice that all the 0s in D_{in} are at the beginning (by the definition of a Wheeler graph), so if D'_{in} is the sequence obtained from D_{in} by deleting its leading 0s, then D'_{in} contains only positive integers, $|D'_{\mathsf{in}}|H_k(D'_{\mathsf{in}}) \leq |D_{\mathsf{in}}|H_k(D_{\mathsf{in}})$ and $D_{\mathsf{in}}.\mathsf{search}(j) = D'_{\mathsf{in}}.\mathsf{search}(j) + |D_{\mathsf{in}}| - |D'_{\mathsf{in}}|$. This gives us our main result:

Theorem 2. *Let G be a Wheeler graph on n vertices with maximum degree Δ. If $\Delta, k \in o\left(\frac{\log n}{(\log\log n)^2}\right)$ then we can store G's out-degree sequence D_{out} in $nH_k(D_{\mathsf{out}}) + o(n)$ bits such that it supports* sum *queries in constant time, and store G's in-degree sequence D_{in} in $nH_k(D_{\mathsf{in}}) + o(n)$ bits such that it supports* search *queries in constant time.*

2 Intuition

The standard approach, proposed by Mäkinen and Navarro [8], to storing a compact searchable partial-sums data structure for a static sequence $S[1..n]$ of positive integers that sum to u, is as a bitvector B in which there are $S[1]-1$ copies of 0 before the first 1 and, for $i > 1$, there are $S[i]-1$ copies of 0 between the $(i-1)$st and ith copies of 1. This takes $n \lg \frac{u}{n} + o(u)$ bits and supports $S.\mathsf{sum}(i) = B.\mathsf{select}_1(i)$ and $S.\mathsf{search}(j) = B.\mathsf{rank}_1(j)$ in constant time. If we use it to store the in- and out-degrees in a BOSS representation of a de Bruijn graph then we use about $\lg \sigma + 2$ bits per edge.

There are many other searchable partial-sums data structures (see, e.g., [3,9] and references therein) but, as far as we know, only a very recent one by Arroyuelo and Raman [1] achieves a space bound in terms of the empirical entropy of S and still answers queries in constant time. It takes $nH_0(S) + O\left(\frac{u(\log\log u)^2}{\log u}\right)$ bits so, if $\max_i\{S[i]\} \in o\left(\frac{\log n}{(\log\log n)^2}\right)$, then $u \in o\left(\frac{n \log n}{(\log\log n)^2}\right)$ and it takes $nH_0(S) + o(n)$ bits. If we apply this instead of Theorem 1 then we obtain a slightly weaker form of Theorem 2, in which H_k is replaced by H_0.

To prove Theorem 1, our starting point is Ferragina and Venturini's [6] well-known result about storing a static string in nH_k-compressed space while supporting fast random access to it:

Theorem 3 (Ferragina and Venturini). *We can store S as a string of n characters from an alphabet of size σ in*

$$nH_k(S) + O\left(\frac{n \log \sigma}{\log n}(k \log \sigma + \log\log n)\right)$$

bits for $k \in o\left(\frac{\log n}{\log \sigma}\right)$ such that we can extract any substring of S of length ℓ in $O\left(1 + \frac{\ell \log \sigma}{\log n}\right)$ time.

Assuming S consists of positive integers with $\max_i\{S[i]\} \in o\left(\frac{\log n}{(\log\log n)^2}\right)$, we have $\sigma \in o\left(\frac{\log n}{(\log\log n)^2}\right)$ and the space bound in Theorem 3 is $nH_k(S) + o(n)$ bits for $k \in o\left(\frac{\log n}{(\log\log n)^2}\right)$. Notice the extraction time is constant for $\ell \in O\left(\frac{\log n}{\log \sigma}\right)$.

In order to support sum and search on S in constant time, we augment Ferragina and Venturini's representation of S with sublinear data structures similar to those Raman, Raman and Rao [10] used to support rank and select on their succinct bitvectors. Since these augmentations are fairly standard, we omit the details of the how we support sum and leave the details of how we support search to the next section.

Lemma 1. *We can add $o(n)$ bits to Ferragina and Venturini's representation of S and support sum in constant time.*

Lemma 2. *We can add $o(n)$ bits to Ferragina and Venturini's representation of S and support* search *in constant time.*

Combining Theorem 3 and Lemmas 1 and 2, we immediately obtain Theorem 1. We note that we need $\sigma \in o\left(\frac{\log n}{(\log\log n)^2}\right)$ only to prove Lemma 2. In the full version of this paper we will show how we can store S in $nH_k(S) + o(n)$ bits of space and support sum queries on it in constant time even when σ is polylogarithmic in n, for example — which could be of interest when storing Wheeler graphs with large maximum out-degree but small maximum in-degree, such as some tries.

3 Proof of Lemma 2

Proof. We first store $\mathsf{search}(c\sigma \lg^2 n)$ for each multiple $c\sigma \lg^2 n$ of $\sigma \lg^2 n$. Since $\mathsf{sum}(n) \leq \sigma n$, this takes a total of

$$O\left(\frac{\sigma n}{\sigma \lg^2 n} \cdot \lg n\right) \subset o(n)$$

bits. We then store the difference

$$\mathsf{search}\left(c \cdot \frac{\lg n}{2 \lg \sigma}\right) - \mathsf{search}\left(\sigma \lg^2(n) \cdot \left\lfloor \frac{c \cdot \frac{\lg n}{2 \lg \sigma}}{\sigma \lg^2 n} \right\rfloor\right)$$

for each multiple $c \cdot \frac{\lg n}{2 \lg \sigma}$ of $\frac{\lg n}{2 \lg \sigma}$ and the preceding multiple $\sigma \lg^2(n) \cdot \left\lfloor \frac{c \cdot \frac{\lg n}{2 \lg \sigma}}{\sigma \lg^2 n} \right\rfloor$ of $\sigma \lg^2 n$. Since each of these differences is at most $\sigma \lg^2 n$, this takes a total of

$$O\left(\frac{\sigma n \log \sigma}{\log n} \cdot \log(\sigma \log^2 n)\right) \subset o(n)$$

bits. Finally, we store a universal table that, for each possible $\frac{\lg n}{2}$-bit encoding of a substring of S consisting of $\frac{\lg n}{2 \lg \sigma}$ integers (each represented by $\lg \sigma$ bits) and each value q between 1 and the maximum possible sum $\sigma \cdot \frac{\lg n}{2 \lg \sigma}$ of such a substring, tells us how many of that substring's integers we can sum before exceeding q. This takes

$$2^{\frac{\lg n}{2} + \lg\left(\sigma \cdot \frac{\lg n}{2 \lg \sigma}\right)} \lg\left(\frac{\lg n}{2 \lg \sigma}\right) \in o(n)$$

bits.

To evaluate $\mathsf{search}(j)$ in constant time, we first look up $\mathsf{search}\left(\sigma \lg^2(n) \cdot \left\lfloor \frac{j}{\sigma \lg^2 n} \right\rfloor\right)$ and

$$\mathsf{search}\left(\frac{\lg n}{2 \lg \sigma} \cdot \left\lfloor \frac{j}{\frac{\lg n}{2 \lg \sigma}} \right\rfloor\right) - \mathsf{search}\left(\sigma \lg^2(n) \cdot \left\lfloor \frac{j}{\sigma \lg^2 n} \right\rfloor\right),$$

which tells us $\mathsf{search}\left(\frac{\lg n}{2\lg\sigma} \cdot \left\lfloor \frac{j}{\frac{\lg n}{2\lg\sigma}} \right\rfloor\right)$. Since

$$j - \frac{\lg n}{2\lg\sigma} \cdot \left\lfloor \frac{j}{\frac{\lg n}{2\lg\sigma}} \right\rfloor < \frac{\lg n}{2\lg\sigma}$$

and the integers in S are positive,

$$\mathsf{search}(j) - \mathsf{search}\left(\frac{\lg n}{2\lg\sigma} \cdot \left\lfloor \frac{j}{\frac{\lg n}{2\lg\sigma}} \right\rfloor\right) < \frac{\lg n}{2\lg\sigma}\,.$$

It follows that we can find $\mathsf{search}(j)$ by extracting the substring of $\frac{\lg n}{2\lg\sigma} \in O\left(\frac{\log n}{\log\sigma}\right)$ characters starting at $S\left[\mathsf{search}\left(\frac{\lg n}{2\lg\sigma} \cdot \left\lfloor \frac{j}{\frac{\lg n}{2\lg\sigma}} \right\rfloor\right)\right]$ and using the universal table to learn how many of that substring's integers we can sum before exceeding

$$j - \mathsf{sum}\left(\mathsf{search}\left(\frac{\lg n}{2\lg\sigma} \cdot \left\lfloor \frac{j}{\frac{\lg n}{2\lg\sigma}} \right\rfloor\right) - 1\right)\,.$$

4 Postscript

We have not implemented Theorems 1 or 2 because there are other approaches that perform poorly in the worst case but are likely unbeatable in practice. If we store S as a degenerate wavelet tree, then we can implement an $S.\mathsf{sum}$ query with σ rank queries on the wavelet trees bitvectors, together with σ multiplications and additions: for example, to find $D_{\mathsf{out}}.\mathsf{sum}(8)$ for the sequence $D_{\mathsf{out}} = 1,1,1,2,1,1,2,1,1,0,1$ with the degenerate wavelet tree shown in Figure 1, we compute

$$1 \cdot B_1.\mathsf{rank}_0(8) + 2 \cdot B_2.\mathsf{rank}_0(8 - B_1.\mathsf{rank}_0(8)) = 1 \cdot 6 + 2 \cdot 2 = 10\,.$$

In practice σ is usually a small constant — often 4 — and if the bitvectors in the wavelet tree are entropy-compressed, then it takes $nH_0(S) + o(n\log\sigma)$ bits. If we store a minimal monotone perfect hash function [2] mapping each value $S.\mathsf{sum}(i)$ to i, together with a small sample of those pairs, then we should also be able to support $S.\mathsf{search}$ queries by computing a few hash values and $S.\mathsf{sum}$ queries, quickly and in small space in practice. We leave the details for the full version of this paper.

Acknowledgments

Many thanks to Jarno Alanko for bringing the topic of this paper to our attention, to Rossano Venturini for pointing out Arroyuelo and Raman's result, and to Meng He, Gonzalo Navarro and Srinivasa Rao Satti for helpful discussions.

References

1. Diego Arroyuelo and Rajeev Raman. Adaptive succinctness. *Algorithmica*, 84:694–718, 2022.
2. Djamal Belazzougui, Paolo Boldi, Rasmus Pagh and Sebastiano Vigna. Theory and practice of monotone minimal perfect hashing. *ACM Journal of Experimental Algorithmics*, 16, 2011.
3. Philip Bille, Inge Li Gørtz and Frederik Rye Skjoldjensen. Partial sums on the ultra-wide word RAM. *Theoretical Computer Science*, 905:99–105, 2022.
4. Christina Boucher, Alex Bowe, Travis Gagie, Simon J. Puglisi and Kunihiko Sadakane. Variable-order de Bruijn graphs. *Proceedings of the Data Compression Conference (DCC)*, pages 383–392, 2015.
5. Alex Bowe, Taku Onodera, Kunihiko Sadakane and Tetsuo Shibuya. Succinct de Bruijn graphs. *Proceedings of the Workshop on Algorithms in Bioinformatics (WABI)*, pages 225–235, 2012.
6. Paolo Ferragina and Rossano Venturini. A simple storage scheme for strings achieving entropy bounds. *Theoretical Computer Science*, 372:115–121, 2007.
7. Travis Gagie, Giovanni Manzini and Jouni Sirén. Wheeler graphs: A framework for BWT-based data structures. *Theoretical Computer Science*, 698:67–78, 2017.
8. Veli Mäkinen and Gonzalo Navarro. Rank and select revisited and extended. *Theoretical Computer Science*, 387:332–347, 2007.
9. Giulio Ermanno Pibiri and Rossano Venturini. Practical trade-offs for the prefix-sum problem. *Software: Practice and Experience*, 51:921–949, 2021.
10. Rajeev Raman, Venkatesh Raman and Srinivasa Rao Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Transactions on Algorithms*, 3:43, 2007.