On the Succinct Representation of Unlabeled Permutations

Hicham El-Zein(✉), J. Ian Munro, and Siwei Yang

Cheriton School of Computer Science, University of Waterloo,
Waterloo, ON N2L 3G1, Canada
{helzein,imunro,siwei.yang}@uwaterloo.ca

Abstract. We investigate the problem of succinctly representing an arbitrary unlabeled permutation π, so that $\pi^k(i)$ can be computed quickly for any i and any integer power k. We consider the problem in several scenarios:

- Labeling schemes where we assign labels to elements and the query is to be answered by just examining the labels of the queried elements: we show that a label space of $\sum_{i=1}^{n} \lfloor \frac{n}{i} \rfloor \cdot i$ is necessary and sufficient. In other words, $2 \lg n$ bits of space are necessary and sufficient for representing each of the labels.
- Succinct data structures for the problem where we assign labels to the n elements from the label set $\{1, \ldots, cn\}$ where $c \geq 1$: we show that $\Theta(\sqrt{n})$ bits are necessary and sufficient to represent the permutation. Moreover, we support queries in such a structure in $O(1)$ time in the standard word-RAM model.
- Succinct data structures for the problem where we assign labels to the n elements from the label set $\{1, \ldots, cn^{1+\epsilon}\}$ where c is a constant and $0 < \epsilon < 1$: we show that $\Theta(n^{(1-\epsilon)/2})$ bits are necessary and sufficient to represent the permutation. We can also support queries in such a structure in $O(1)$ time in the standard word-RAM model.

1 Introduction and Motivation

A permutation π is a bijection from the set $\{1, \ldots, n\}$ to itself. Given a permutation π on an n element set, our problem is to preprocess the set, assigning a unique label to each element, to obtain a data structure with minimum space to support the following query: given a label i, determine $\pi^k(i)$ quickly. We denote such queries by $\pi^k()$. Moreover, we assume that k is bounded by some polynomial function in n.

We are interested in *succinct*, or highly-space efficient data structures. Our aim is to develop data structures whose size is within a constant factor of the information theoretic lower bound. Designing succinct data structures is an area of interest in theory and practice motivated by the need of storing large amount

This work was sponsored by the NSERC of Canada and the Canada Research Chairs Program.

K. Elbassioni and K. Makino (Eds.): ISAAC 2015, LNCS 9472, pp. 49–59, 2015.
DOI: 10.1007/978-3-662-48971-0_5

of data using the smallest space possible. For succinct representations of dictionaries, trees, arbitrary graphs, partially ordered sets and equivalence relations see [1,3,5,6,11,12,14].

Permutations are fundamental in computer science and are studied extensively. Several papers have looked into problems related to permutation generation [15], permuting in place [7] etc. Others have dealt with the problem of space-efficient representation of restricted classes of permutations, like the permutations representing the lexicographic order of the suffixes of a string [8,10], or the so-called approximately min-wise independent permutations [2], which are used for document similarity estimation. Since there are exactly $n!$ permutations, the number of bits required to represent a permutation of length n is $\lceil \lg(n!) \rceil \sim n \lg n - n \lg e + O(\lg n)$[1] bits. Munro et al. [13] studied the space efficient representation of general permutations where general powers can be computed quickly. They gave a representation taking the optimal $\lceil \lg(n!) \rceil + o(n)$ bits, and a representation taking $((1+\epsilon)n \lg n)$ bits where $\pi^k()$ can be computed in constant time.

Our paper is the first to study the space-efficient representation of permutations where labels can be freely reassigned. This problem is similar to the problem of representing unlabeled equivalence relations [5,11]. However, our problem differs from representing equivalence relations when the label space exceeds n. In our case we must know the size of each cycle, while for equivalence relations it is not necessary to know the exact size of the equivalence classes. Thus, as we increase the label space we will not witness a drastic decrease in auxiliary storage size. We study this problem in several scenarios; thus, showing the tradeoffs between label space and auxiliary storage size for the stated problem. In Sect. 3, we cover the scenario where queries are to be answered by just examining the labels of the queried elements. We show that a label space of $\sum_{i=1}^{n} \lfloor \frac{n}{i} \rfloor \cdot i$ is necessary and sufficient. Then, we show that with a label space of n^2 queries can be answered in constant time. In Sect. 4, we cover the scenario where labels can be assigned from the set $\{1, \ldots, n\}$. We show that $\Theta(\sqrt{n})$ bits are necessary and sufficient to represent the permutation. We use the same data structure as the main structure in [11]. However, we optimize it to achieve constant query time while using only $O(\sqrt{n})$ bits; thus, solving an open problem from [11]. Note that the details of this improvement are also found in the first author's thesis [4]. Section 5 contains the main result of this paper. We cover the scenario where labels can be assigned from the set $\{1, \ldots, cn^{1+\epsilon}\}$ where c is a constant and $0 < \epsilon < 1$. We show that $\Theta(n^{(1-\epsilon)/2})$ bits are necessary and sufficient to represent the permutation, and we support queries in such a structure in $O(1)$ time in the standard word-RAM model.

Finally as an application to our new data structures, we give a representation of a labeled permutation that takes $s(n) + O(\sqrt{n})$ bits and can answer $\pi^k()$ in $O(t_f + t_i)$ time, where $s(n)$ denotes the number of bits required for a representation R to store a labeled permutation, and t_f and t_i are the time needed for R to support $\pi()$ and $\pi^{-1}()$. This result improves Theorem 3.3 in [13].

[1] We use $\lg n$ to denote $\log_2 n$.

2 Definitions and Preliminaries

A *permutation* π is a bijection from the set $\{1, \ldots, n\}$ to itself, and we denote its inverse bijection as π^{-1}. We also extend the definition to arbitrary integer power of π as follows:

$$\pi^k(i) = \begin{cases} \pi^{k+1}(\pi^{-1}(i)) & k < 0 \\ i & k = 0 \\ \pi^{k-1}(\pi(i)) & k > 0 \end{cases}$$

A permutation can be viewed as a set of disjoint cycles. Since we are working with unlabeled permutations, we have the freedom to assign the labels in any way. In all our labeling schemes, we give elements within the same cycle and cycles of the same length consecutive labels. For example the elements of the first cycle of length l will get labels from the interval $[s, s + l - 1]$, such that $\pi(i) = i + 1$ for $i \in [s, s + l - 2]$ and $\pi(s + l - 1) = s$. The elements of the second cycle of length l will get labels in the range $[s + l, s + 2l - 1]$, and so on. Thus given a label i and an integer k, to answer $\pi^k(i)$ it is sufficient to compute l the length of the cycle that i belongs to, and s the smallest index of an element that belongs to a cycle of length l. Now, it is not hard to verify that $\pi^k(i) = s + rl + ((p + k)\%l)$[2] where $r = \lfloor (i - s)/l \rfloor$ and $p = i - (s + rl)$.

Notice that the multiset formed by the cycles lengths of a given permutation π over an n-element set will form an integer partition of the integer n. An *integer partition* p of n is a multiset of positive integers that sum to n. We call these positive integers the *elements* of p, and we denote by $|p|$ this number of elements. We say that an integer partition p of n *dominates* an integer partition q of m where $n > m$ if q is a subset of p. For example, the integer partition $\{5, 5, 10\}$ of 20 dominates the integer partition $\{5, 5\}$ of 10, but not the integer partition $\{4, 6\}$ of 10. Given an integer partition p of n, we define a *part* q of size k to be a collection of elements in p that sum to k. We say that an integer s *fills* q if q contains $\lfloor k/s \rfloor$ integers s and one integer $k \bmod s$. Furthermore, we say that two parts *intersect* if they share at least one common element; otherwise, they are *non-intersecting*. For example the integer partition $\{1, 4, 5\}$ of 10 contains the following parts: part $\{1\}$ of size 1, part $\{4\}$ of size 4, part $\{5\}$ of size 5, part $\{1, 4\}$ of size 5, part $\{1, 5\}$ of size 6, part $\{4, 5\}$ of size 9 and part $\{1, 4, 5\}$ of size 10. We say that 5 fills the parts $\{5\}$ and $\{4, 5\}$ but not the part $\{1, 4, 5\}$. The parts $\{4, 5\}$ and $\{4\}$ are intersecting, while the parts $\{4, 5\}$ and $\{1\}$ are non-intersecting.

Finally, we give two observations that we will use repeatedly.

Observation 1. *M not necessarily distinct integers $m_0, \ldots, m_{M-1}$ ordered such that $m_i \leq m_{i+1}$ in the range $[0, N-1]$,can be represented in $O(N+M)$ bits such that the i^{th} integer m_i can be accessed in $O(1)$ time.*

[2] We use % to denote the modulo operation.

Observation 2. *M positive integers $m_0, \ldots, m_{M-1}$ that sum to N can be represented in $O(N+M)$ bits such that the i^{th} integer m_i can be accessed in $O(1)$ time, the partial sum $\sum_{j=1}^{i} m_j$ can be computed in $O(1)$ time, and given an integer x we can compute the biggest index i such that $\sum_{j=1}^{i} m_j \leq x$ in $O(1)$ time.*

The proof of both observations is found in the appendix. Note that if we are allowed to reorder the numbers in Observation 2, we can reduce the size of the representation to $O(\sqrt{N})$ bits without compromising the constant runtime of the stated operations.

3 Direct Labeling Scheme

In this section we cover the problem where queries are answered by computing directly from the labels without using any auxiliary storage except for the value of n. We show that a label space of $\sum_{i=1}^{n} \lfloor \frac{n}{i} \rfloor \cdot i$ is necessary and sufficient to represent the permutation. Moreover, we show that with a label space of n^2 $\pi^k()$ can be computed in constant time.

Theorem 3. *Given a permutation π, a label space of $\sum_{i=1}^{n} \lfloor \frac{n}{i} \rfloor \cdot i$ is necessary and sufficient to represent the permutation.*

For the proof of Theorem 3 check the appendix. To answer queries in constant time we extend the label space to n^2. Then we assign labels from the set of integers in the range $[0, n-1]$ for all the elements in cycles of length 1, and labels from the set of integers in the range $[n(i-1)+(r-1)i, n(i-1)+ri-1]$ for the elements in the r^{th} cycle of length i, where $1 \leq r \leq \lfloor n/i \rfloor$. Given a label x, to answer a query $\pi^k(x)$ find $l = \lfloor x/n \rfloor + 1$. Next, compute $s = (l-1)n$, $r = \lfloor (x-s)/l \rfloor$ and $p = x - (s + rl)$, then return $s + rl + ((p+k)\%l)$.

Theorem 4. *Given a permutation π, we can assign to each of the elements a label in the range of $\{1, \ldots, n^2\}$ such that $\pi^k()$ can be computed in constant time by looking only at the labels.*

4 Succinct Data Structures with Label Space n

In this section we consider the scenario where the n elements are to be assigned labels in the range 1 to n. The queries can be answered by looking at an auxiliary data structure. Moreover, we have the freedom to assign the labels in any way.

Following [11], the information theoretic lower bound for the representation of a permutation is the number of partitions of n, which by the Hardy-Ramanujan formula [9] is asymptotically equivalent to $\frac{1}{4n\sqrt{3}} e^{\pi\sqrt{\frac{2n}{3}}}$. Thus the information theoretic lower bound for representing a permutation is $\Theta(\sqrt{n})$ bits of space.

We will use the same data structure as the main structure in [11], however we will optimize it to achieve constant query time while using only $O(\sqrt{n})$ bits. Given π let k be the number of distinct cycle sizes in π. For $i = 1$ to k, let s_i

be the distinct sizes of the cycles, and let n_i be the number of cycles of size s_i. Order the cycles in non-decreasing order by $\gamma_i = s_i n_i$ so that for $i = 1$ to $k-1$, $s_i n_i \leq s_{i+1} n_{i+1}$. Notice that since

$$\sum_{i=1}^{k} s_i n_i = n \text{ and } s_i n_i \geq i \text{ for } i = 1, \ldots, k, \tag{1}$$

k is at most $\sqrt{2n}$. The primary data structure is made up of two sequences:

- the sequence $\boldsymbol{\delta}$ that consists of $\delta_1 = s_1 n_1$ and $\delta_i = s_i n_i - s_{i-1} n_{i-1}$, for $i = 2, \ldots, k$ and
- the sequence $\boldsymbol{n}$ that consists of n_i, for $i = 1, \ldots, k$.

Elements of the two sequences are represented in binary. Since the length of each element may vary, we store two other sequences that shadow the primary sequences. The shadow sequences have a 1 at the starting point of each element in the shadowed sequence and a 0 elsewhere. Also store a select structure on the two shadow sequences in order to identify the 1s quickly. It is proved in [11] these sequences can be stored in $O(\sqrt{n})$ bits.

The sequence $\boldsymbol{\delta}$ gives an implicit ordering of the elements. Assign the first $s_1 n_1$ labels to the elements of the cycles with length s_1, the elements of the next n_2 cycles are assigned the next $s_2 n_2$ labels and so on.

Denote by the predecessor of an element x to be $\max\{j \mid \sum_{i=1}^{j} s_i n_i < x\}$. Store an array A, where $A[i] = \max\{j \mid \sum_{t=1}^{j} s_t n_t \leq i(i+1)/2\}$, for $i = 1$ to $\sqrt{2n}$. Next, we prove a modified version of Lemma 2 in [11].

Lemma 1. *The predecessor $p(x)$ of an integer x in the sequence $\sum_{t=1}^{i} s_t n_t$, $i = 1$ to k is in the range $[A[\lfloor\sqrt{2x}\rfloor - 1], A[\lfloor\sqrt{2x}\rfloor - 1] + 5]$.*

Proof. Let $i = \lfloor\sqrt{2x}\rfloor - 1$. Without loss of generality assume that $i \geq 6$, since for $x < 25$ we can store $p(x)$ explicitly in $O(\lg n)$ bits. Notice that

$$i(i+1)/2 \leq (\sqrt{2x} - 1)\sqrt{2x}/2 \leq x$$

and

$$x \leq \sqrt{2x}(\sqrt{2x} + 1)/2 \leq (i+2)(i+3)/2$$

For $j = A[i] + 1$, $\sum_{t=1}^{j-1} s_t n_t \leq i(i+1)/2$, so $j - 1 \leq i$ and $j \leq i + 1$. Since $\sum_{t=1}^{j} s_t n_t > i(i+1)/2$, $s_j n_j \geq i(i+1)/(2j) \geq i/2$. Hence, $\sum_{t=1}^{j+5} s_t n_t \geq (i+2)(i+3)/2 \geq x$. □

The actual value of $p(x)$ can be obtained by checking at most six numbers. Moreover, A can be stored using $O(\sqrt{n})$ bits using the method described in Observation 1.

In the standard word-RAM model, computing $\sqrt{x}$ is not a constant time operation. The standard Newton's iterative method uses $O(\lg \lg n)$ operations. Following [11], we can use a look-up to precomputed tables and finds $\sqrt{x}$ in

constant time. We use two tables, one when the number of bits up to the most significant bit of x is odd, denoted by O, and one when the number of bits is even, denoted by E. For $i = 1, \ldots, \lceil\sqrt{2n}\rceil$, we store in $E[i]$ the value of $\lfloor\sqrt{i2^{\lceil \lg i\rceil}}\rfloor$, and in $O[i]$ the value of $\lfloor\sqrt{i2^{\lceil \lg i\rceil - 1}}\rfloor$. E and O can be stored in $O(\sqrt{n})$ bits by storing them using the method described in Observation 1.

Lemma 2. *For $i \leq n$, $\lfloor\sqrt{i}\rfloor$ can be computed in constant time using a precomputed table of $O(\sqrt{n})$ bits.*

For each i, where at least one of δ_i's bits locations in $\boldsymbol{\delta}$ is a multiple of $(\epsilon \lg n)$, store the partial sum value $\sum_{j=1}^{i}(s_j n_j)$ and the value of $s_i n_i$. Moreover, for every possible sequence of δ values $\delta_1, \delta_2, \ldots, \delta_i$ of length $(\epsilon \lg n)$ and its corresponding shadow sequence, store in a table T the values i and $\sum_{j=1}^{i}(\sum_{k=1}^{j}\delta_k)$. To compute $\sum_{j=1}^{i}(s_j n_j)$ for an arbitrary index i, find the biggest index $k \leq i$ that has it's partial sum value stored. Notice that $\sum_{j=1}^{i}(s_j n_j) = \sum_{j=1}^{k}(s_j n_j) + (i-k)s_k n_k + \sum_{j=k+1}^{i}(\sum_{l=k+1}^{i}\delta_l)$. Since these values can be obtained using table lookup on T, we can compute the partial sum at an arbitrary index in constant time. Moreover, we can compute the value of $s_i n_i$ for an arbitrary index i by computing the partial sum at $i-1$ and subtracting it from the partial sum at i. Finally, we can compute s_i by computing $s_i n_i$ and dividing it by n_i. By choosing $\epsilon < 1/4$, the size of T becomes $o(\sqrt{n})$ bits.

Answering Queries: Given a label x, to compute $\pi^k(x)$ we first find the predecessor $p(x)$ of x by querying A and checking at most 6 different values. Next we compute the partial sum value $s = \sum_{i=1}^{p(x)-1}(n_i s_i)$. Then, compute $r = \lfloor (x-s)/s_{p(x)}\rfloor$ and $p = x - (s + r s_{p(x)})$, then return $s + r s_{p(x)} + ((p+k)\%l)$.

Theorem 5. *Given an unlabeled permutation of n elements, $\Theta(\sqrt{n})$ bits are necessary and sufficient for storing the permutation if each element is to be given a unique label in the range $\{1, 2, \ldots, n\}$. Moreover, $\pi^k()$ can be computed in $O(1)$ time in such a structure.*

5 Succinct Data Structures with Label Space $cn^{1+\epsilon}$

In this section we consider the scenario where the n elements are to be assigned labels in the range 1 to $cn^{1+\epsilon}$ where c is a constant and $0 < \epsilon < 1$. As in Sect. 4 we assign an implicit ordering of the elements, and queries can be answered by looking at an auxiliary data structure.

Given π, we divide the cycles in π into four different groups and handle each group appropriately. For $i = 1$ to k_3, let s_i be the distinct sizes of the cycles of size $\leq n^{(1+\epsilon)/2}$, and let n_i be the number of cycles of size s_i. Without loss of generality, assume that:

- $\gamma_i = s_i n_i \leq (\sqrt{c}n^{(1+\epsilon)/2})/2 = \eta$, for $1 \leq i \leq k_1$.
- $s_i \leq n^{(1-\epsilon)/2}$ and $\gamma_i > \eta$, for $k_1 < i \leq k_2$.

– $n^{(1-\epsilon)/2} < s_i \leq n^{(1+\epsilon)/2}$ and $\gamma_i > \eta$, for $k_2 < i \leq k_3$.

Let $l_{k_3+1}, \ldots, l_{k_4}$ be the size of the cycles that are bigger than $n^{(1+\epsilon)/2}$. Note that the l_i ($i = k_3 + 1$ to k_4) values are not necessarily unique.

Case 1: Reserve the first $(cn^{1+\epsilon})/4$ labels to handle all possible cycle sizes when $\gamma_i \leq \eta$. Assign labels to the elements in the cycles that satisfy this criteria in a similar method to the labeling scheme described in Theorem 4. To be more specific, we assign labels from the set of integers in the range $[0, \eta - 1]$ for all the elements in cycles of length 1, and assign labels from the set of integers in the range $[\eta(j-1), \eta j - 1]$ for all the elements in cycles of length j, where $2 \leq j \leq \eta$. This covers all the elements of the cycles of sizes $s_1, \ldots, s_{k_1}$, and increases the label space by at most $\eta^2 = (cn^{1+\epsilon})/4$. Let $B_1 = (cn^{1+\epsilon})/4$.

Case 2 ($k_1 + 1 \leq i \leq k_2$): Order the s_i values in increasing order. Make sure that all cycles of size s_i, fill a part whose length is $c_i\eta$ a multiple of η. Notice that $(k_2 - k_1) < n/\eta$ since $\gamma_i > \eta$, so the label space will increase by at most n. Since $\sum_{i=k_1+1}^{k_2}(c_i) \leq (2n)/\eta = O(n^{(1-\epsilon)/2})$, we can store the c_i values in $O(n^{(1-\epsilon)/2})$ bits using the method described in Observation 2. Moreover, we store a bit vector ψ of size $n^{(1-\epsilon)/2}$ to identify the s_i values, and we store a select structure on ψ to identify the $1s$ quickly. Assign labels in the range $[B_1, B_1 + c_{(k_1+1)}\eta - 1]$ to the elements in cycles of size $s_{(k_1+1)}$, then assign the next $c_{(k_1+2)}\eta$ labels to elements in cycles of size $s_{(k_1+2)}$, and so on. Let $B_2 = B_1 + \sum_{j=k_1+1}^{k_2} c_j\eta$.

Case 3 ($k_2 + 1 \leq i \leq k_3$): Make sure that all cycles of size s_i, fill a part whose length is $c_i\eta$ a multiple of η. As in case 2, store the c_i values in $O(n^{(1-\epsilon)/2})$ bits using the method described in Observation 2. To identify the s_i values: order them in increasing order of $r_i = s_i \% (16n^{(1-\epsilon)/2}/c)$ and store the r_i values in $O(n^{(1-\epsilon)/2})$ bits using the method described in Observation 1, then store the value of $q_i = s_i/(16n^{(1-\epsilon)/2}/c) \leq (cn^{\epsilon}/16)$ in the label of each element that is in a cycle of size s_i. Now $s_i = q_i(16n^{(1-\epsilon)/2}/c) + r_i$. Let β_1 be equal to $\sum_{i=k_2+1}^{k_3} c_i\eta$. Assign labels in the range

$$\left[B_2 + q_i 2^{\lceil \lg(\beta_1) \rceil} + \sum_{j=k_2+1}^{i-1} c_j\eta, B_2 + q_i 2^{\lceil \lg(\beta_1) \rceil} + \sum_{j=k_2+1}^{i} c_j\eta - 1\right]$$

to the elements in the cycles of size s_i. The label space will increase by at most $(cn^{\epsilon}/16)2^{\lceil \lg(\beta_1) \rceil} + \beta_1 \leq (cn^{1+\epsilon})/4 + O(n)$. Let $B_3 = B_2 + (cn^{\epsilon}/16)2^{\lceil \lg(\beta_1) \rceil} + \beta_1$.

Case 4 ($k_3 + 1 \leq i \leq k_4$): For the cycles of length l_i, make sure that each cycle fills a part whose length is $c_i\eta$ a multiple of η. As in the previous cases, store the c_i values in $O(n^{(1-\epsilon)/2})$ bits using the method described in Observation 2. To identify the l_i values: order them by $r_i = (l_i \% \eta) \% (8n^{(1-\epsilon)/2}/\sqrt{c})$ and store the r_i values in $O(n^{(1-\epsilon)/2})$ bits using the method described in Observation 1, then store the value of $q_i = (l_i \% \eta)/(8n^{(1-\epsilon)/2}/\sqrt{c}) \leq (cn^{\epsilon}/16)$ in the label of each element that is in a cycle of size l_i. Now $l_i = q_i(8n^{(1-\epsilon)/2}/\sqrt{c}) + r_i + (c_i - 1)\eta$. Let β_2 be equal to $\sum_{i=k_3+1}^{k_4} c_i\eta$. Assign labels in the range

$$\left[B_3 + q_i 2^{\lceil \lg(\beta_2) \rceil} + \sum_{j=k_3+1}^{i-1} c_j\eta, B_3 + q_i 2^{\lceil \lg(\beta_2) \rceil} + \sum_{j=k_3+1}^{i} c_j\eta - 1\right]$$

to the elements in the cycle of size l_i.

The total size of the structures used is $O(n^{(1-\epsilon)/2})$ bits, and the total address space increased to at most $(3cn^{1+\epsilon})/4 + O(n) \leq cn^{1+\epsilon}$ as required.

Answering Queries: Given a label x, to compute $\pi^k(x)$ we distinguish between four different cases:

Case 1 $x < B_1$**:** Compute the value of $l = \lfloor x/\eta \rfloor + 1$, $s = (l-1)\eta$, $r = \lfloor (x-s)/l \rfloor$, and $p = x - (s + rl)$. Then, return $s + rl + ((p + k)\%l)$.

Case 2 $B_1 \leq x < B_2$**:** Compute the value $m = (x - B_1)/\eta$. Then get the biggest index i such that $\sum_{j=k_1+1}^{i} c_j \leq m$. This operation can be done in $O(1)$ time using the structure from Observation 2. Next, find l the index of the i^{th} one in ψ; l is the size of the cycle that x belongs to. Compute $s = B_1 + \sum_{j=k_1+1}^{i-1} c_j\eta$, $r = \lfloor (x - s)/l \rfloor$, and $p = x - (s + rl)$. Then, return $s + rl + ((p + k)\%l)$.

Case 3 $B_2 \leq x < B_3$**:** Compute the value $m = ((x - B_2)\%\beta_1)/\eta$. Then get the biggest index i such that $\sum_{j=k_2+1}^{i} c_j \leq m$. Next calculate $q_i = \lfloor (x - B_2)/2^{\lceil \lg(\beta_1) \rceil} \rfloor$ and $l = q_i(16n^{(1-\epsilon)/2}/c) + r_i$; l is the size of the cycle that x belongs to. Compute $s = B_2 + q_i 2^{\lceil \lg(\beta_1) \rceil} + \sum_{j=k_2+1}^{i-1} c_j\eta$, $r = \lfloor (x - s)/l \rfloor$, and $p = x - (s + rl)$. Then, return $s + rl + ((p + k)\%l)$.

Case 4 $B_3 \leq x$**:** Compute the value $m = ((x - B_3)\%\beta_2)/\eta$. Then get the biggest index i such that $\sum_{j=k_3+1}^{i} c_j \leq m$. Next calculate $q_i = \lfloor (x - B_3)/2^{\lceil \lg(\beta_2) \rceil} \rfloor$ and $l = q_i(8n^{(1-\epsilon)/2}/\sqrt{c}) + r_i + (c_i - 1)\eta$; l is the size of the cycle that x belongs to. Compute $s = B_3 + q_i 2^{\lceil \lg(\beta_2) \rceil} + \sum_{j=k_3+1}^{i-1} c_j\eta$, $r = \lfloor (x-s)/l \rfloor$, and $p = x-(s+rl)$. Then, return $s + rl + ((p + k)\%l)$.

All operations used take constant time, so $\pi^k(x)$ can be computed in $O(1)$ time.

Theorem 6. *Given an unlabeled permutation of n elements, $\Theta(n^{(1-\epsilon)/2})$ bits are sufficient for storing the permutation if each element is to be given a unique label in the range $\{1, \ldots, cn^{1+\epsilon}\}$ for any constant $c > 1$ and $\epsilon < 1$. Moreover, $\pi^k()$ can be computed in $O(1)$ time in such a structure.*

Note that ϵ doesn't need to be a constant. By setting $\epsilon = \alpha + \beta \lg\lg n/\lg n$ where α and β are constants, and $0 < \alpha < 1$ we get the following theorem:

Theorem 7. *Given an unlabeled permutation of n elements, $\Theta(n^{(1-\alpha)/2}/\lg^{\beta/2} n)$ bits are sufficient for storing the permutation if each element is to be given a unique label in the range $\{1, \ldots, cn^{1+\alpha} \lg^{\beta} n\}$ for any constant c, α, β where $0 < \alpha < 1$. Moreover, $\pi^k()$ can be computed in $O(1)$ time in such a structure.*

6 Lower Bounds

In this section we provide lower bounds on the auxiliary data size as the label space increases.

6.1 Lower Bound for Auxiliary Data with Label Space *cn*

In [5] El-Zein et al. showed that for the problem of representing unlabeled equivalence relations, increasing the label space by a constant factor causes the size of the auxiliary data structure to decrease from $O(\sqrt{n})$ to $O(\lg n)$ bits.

In contrast to the problem of representing unlabeled equivalence relations, in this section we show that for the problem of representing unlabeled permutations increasing the label space by a constant factor will not affect the size of the auxiliary data structure asymptotically.

For any integer $c > 1$, let S_{cn} be the set of all partitions of $\lfloor cn \rfloor$ and S_n the set of all partitions of n. Without loss of generality assume that $\sqrt{n}$ is an integer that is divisible by c. While one partition of cn can dominate many partitions of n, we argue that at least $\binom{c\sqrt{n}}{\sqrt{n}/c}/\binom{\sqrt{n}}{\sqrt{n}/c}$ partitions of cn are necessary to dominate all partitions of n. Let $\mathcal{S}$ be the smallest set of partitions of cn that dominates all the partitions of n. We claim that:

Lemma 3. $|\mathcal{S}| \geq \binom{c\sqrt{n}}{\sqrt{n}/c}/\binom{\sqrt{n}}{\sqrt{n}/c}$. *The proof of Lemma 3 is found in the appendix. The information theoretic lower bound for the space needed to represent a permutation of size n once labels are assigned from the set* $\{1, \ldots, cn\}$ *is*

$$\begin{aligned}\lg(|\mathcal{S}|) &\geq \lg\left(\binom{c\sqrt{n}}{\sqrt{n}/c}/\binom{\sqrt{n}}{\sqrt{n}/c}\right)\\ &\in \Omega(\sqrt{n}).\end{aligned}$$

Theorem 8. *Given an unlabeled permutation of n elements,* $\Theta(\sqrt{n})$ *bits are necessary and sufficient for storing the permutation if each element is to be given a unique label in the range* $\{1, \ldots, cn\}$ *for any constant* $c > 1$. *Moreover,* $\pi^k()$ *can be computed in* $O(1)$ *time in such a structure.*

6.2 Lower Bound for Auxiliary Data with Label Space $cn^{1+\epsilon}$

Using techniques that are similar to the techniques presented in the previous subsection, we show that for the problem of representing unlabeled permutations an auxiliary data structure of size $O(n^{(1-\epsilon)/2})$ bits is necessary when the label space is $cn^{1+\epsilon}$, where c is any constant and $0 < \epsilon < 1$.

Denote by $S_{cn^{1+\epsilon}}$ the set of all partitions of $cn^{1+\epsilon}$ and by S_n the set of all partitions of n. We argue that at least $\binom{(c+1)n^{(1+\epsilon)/2}}{n^{(1-\epsilon)/2}/(c+1)}/\binom{cn^{(1+\epsilon)/2}/(c+1)}{n^{(1-\epsilon)/2}/(c+1)}$ are necessary to dominate all partitions of n. Let $\mathcal{S}$ be the smallest set of partitions of $cn^{1+\epsilon}$ that dominates all partitions of n. We claim that:

Lemma 4. $|\mathcal{S}| \geq \binom{(c+1)n^{(1+\epsilon)/2}}{n^{(1-\epsilon)/2}/(c+1)} / \binom{cn^{(1+\epsilon)/2}/(c+1)}{n^{(1-\epsilon)/2}/(c+1)}$. *The proof of Lemma 4 is found in the appendix. The information theoretic lower bound for space to represent a permutation of size n once labels are assigned from the set* $\{1, \ldots, cn^{1+\epsilon}\}$ *is*

$$\begin{aligned} \lg(|\mathcal{S}|) &\geq \lg\left(\binom{(c+1)n^{(1+\epsilon)/2}}{n^{(1-\epsilon)/2}/(c+1)} / \binom{cn^{(1+\epsilon)/2}/(c+1)}{n^{(1-\epsilon)/2}/(c+1)}\right) \\ &\in \Omega(n^{(1-\epsilon)/2}). \end{aligned}$$

Theorem 9. *Given an unlabeled permutation of n elements,* $\Theta(n^{(1-\epsilon)/2})$ *bits are necessary and sufficient for storing the permutation if each element is to be given a unique label in the range* $\{1, \ldots, cn^{1+\epsilon}\}$ *for any constant* $c > 1$ *and* $\epsilon < 1$. *Moreover,* $\pi^k()$ *can be computed in* $O(1)$ *time in such a structure.*

7 Applications

As an application to our data structures, we give a representation of a labeled permutation that takes $s(n) + O(\sqrt{n})$ bits and can answer $\pi^k()$ in $O(t_f + t_i)$ time, where $s(n)$ denotes the number of bits required for a representation R to store a labeled permutation, and t_f and t_i are the time needed for R to support $\pi()$ and $\pi^{-1}()$.

This result improves Theorem 3.3 in [13]: Suppose there is a representation R taking $s(n)$ bits to store an arbitrary permutation π on $\{1, \ldots, n\}$, that supports $\pi()$ in time t_f, and $\pi^{-1}()$ in time t_i. Then there is a representation for an arbitrary permutation on $\{1, \ldots, n\}$ taking $s(n) + O(n \lg n / \lg \lg n)$ bits in which $\pi^k()$ can be supported in $t_f + t_i + O(1)$ time, and one taking $s(n) + O(\sqrt{n} \lg n)$ bits in which $\pi^k()$ can be supported in $t_f + t_i + O(\lg \lg n)$ time.

Theorem 10. *Suppose there is a representation R taking* $s(n)$ *bits to store an arbitrary permutation* π *on* $\{1, \ldots, n\}$, *that supports* $\pi()$ *and* $\pi^{-1}()$ *in time* t_f *and* t_i. *Then there is a representation for an arbitrary permutation on* $\{1, \ldots, n\}$ *taking* $s(n) + O(\sqrt{n})$ *bits in which* $\pi^k()$ *can be supported in* $t_f + t_i + O(1)$ *time.*

Proof. Given π, treat it as an unlabeled permutation and build the data structure from Theorem 5 on it. Call this structure P. Notice that the bijection between the labels generated by P and the real labels of π form a permutation. Store this permutation using the given scheme in a structure P'. Now $\pi^k(i) = \pi_{P'}^{-1}(\pi_P^k(\pi_{P'}^1(i)))$ can be computed in $t_f + t_i + O(1)$ time, and the total space used is $s(n) + O(\sqrt{n})$ bits. □

8 Conclusion

We have provided a complete breakdown for the label space-auxiliary storage size tradeoff for the problem of representing unlabeled permutations. As there is a huge body of research in 'labeling schemes', investigation into such a tradeoff for other problems maybe interesting. Moreover as an application to our new data structures, we showed how to improve the general representation of permutations. Given that permutations are fundamental in computer science, we feel that our structures will find applications in many other scenarios.

References

1. Barbay, J., Aleardi, L.C., He, M., Munro, J.I.: Succinct representation of labeled graphs. Algorithmica **62**(1–2), 224–257 (2012)
2. Broder, A.Z., Charikar, M., Frieze, A.M., Mitzenmacher, M.: Min-wise independent permutations. J. Comput. Syst. Sci. **60**(3), 630–659 (2000)
3. Brodnik, A., Munro, J.I.: Membership in constant time and almost-minimum space. SIAM J. Comput. **28**(5), 1627–1640 (1999)
4. El-Zein, H.: On the succinct representation of equivalence classes (2014)
5. El-Zein, H., Munro, J.I., Raman, V.: Tradeoff between label space and auxiliary space for representation of equivalence classes. In: Ahn, H.-K., Shin, C.-S. (eds.) ISAAC 2014. LNCS, vol. 8889, pp. 543–552. Springer, Heidelberg (2014)
6. Farzan, A., Munro, J.I.: Succinct representations of arbitrary graphs. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 393–404. Springer, Heidelberg (2008)
7. Fich, F.E., Munro, J.I., Poblete, P.V.: Permuting in place. SIAM J. Comput. **24**(2), 266–278 (1995)
8. Grossi, R., Vitter, J.S.: Compressed suffix arrays and suffix trees with applications to text indexing and string matching. SIAM J. Comput. **35**(2), 378–407 (2005)
9. Hardy, G.H., Ramanujan, S.: Asymptotic formulae in combinatory analysis. Proc. London Math. Soc. **2**(1), 75–115 (1918)
10. He, M., Munro, J.I., Rao, S.S.: A categorization theorem on suffix arrays with applications to space efficient text indexes. In: Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 23–32. SIAM (2005)
11. Lewenstein, M., Munro, J.I., Raman, V.: Succinct data structures for representing equivalence classes. In: Cai, L., Cheng, S.-W., Lam, T.-W. (eds.) ISAAC 2013. LNCS, vol. 8283, pp. 502–512. Springer, Heidelberg (2013)
12. Munro, J.I., Nicholson, P.K.: Succinct posets. In: Epstein, L., Ferragina, P. (eds.) ESA 2012. LNCS, vol. 7501, pp. 743–754. Springer, Heidelberg (2012)
13. Munro, J.I., Raman, R., Raman, V., Rao, S.S.: Succinct representations of permutations and functions. Theoret. Comput. Sci. **438**, 74–88 (2012)
14. Raman, R., Raman, V., Satti, S.R.: Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. ACM Trans. Algorithms **3**(4) (2007). Article no 43
15. Sedgewick, R.: Permutation generation methods. ACM Comput. Surv. **9**(2), 137–164 (1977)