Succinct Representations of Permutations

J. Ian Munro[1], Rajeev Raman[2], Venkatesh Raman[3], and S. Srinivasa Rao[1]

[1] School of Computer Science, University of Waterloo, Waterloo ON, Canada N2L 3G1. imunro@uwaterloo.ca, ssrao@uwaterloo.ca.

[2] Dept. of Maths & CS, Univ. of Leicester, Leicester LE1 7RH, UK. rr29@le.ac.uk.

[3] Institute of Mathematical Sciences, Chennai, India 600 113. vraman@imsc.res.in

Abstract. We investigate the problem of succinctly representing an arbitrary permutation, π, on n elements so that $\pi^k(i)$ can still be computed quickly for any element i and any (positive or negative integer) power k. A representation taking $(1+\epsilon)n \lg n$ bits suffices to compute arbitrary powers in constant time. A representation taking the optimal $\lceil \lg n! \rceil + o(n)$ bits can be used to compute arbitrary powers in $O(\lg n / \lg\lg n)$ time, and indeed a minimal $O(\lg n)$ bit probes.

1 Introduction

We consider the problem of representing permutations (abbreviated hereafter as *perms* [7]) of $[n] = \{0, \ldots, n-1\}$. Perms are fundamental in computer science and have been the focus of extensive study. A number of papers have dealt with issues pertaining to perm generation, membership in perm groups etc. Our aim here is to develop a "perm data structure" that is, we are given a specific and arbitrary (static) perm that arises in some application, and have to represent this perm so that operations on it can be performed rapidly. Initially motivated by being able to compute π or π^{-1} quickly, we consider the more general operation of computing $\pi^k(i)$ for any integer k, where $\pi^0(i) = i$ for all i; $\pi^k(i) = \pi(\pi^{k-1}(i))$ when $k > 0$ and $\pi^k(i) = \pi^{-1}(\pi^{k+1}(i))$ when $k < 0$.

Certainly, for static perms the above problem is trivial if space is not an issue. Our interest here is in *succinct* or very space-efficient representations that approach the information-theoretic lower bound of $\mathcal{P}(n) = \lceil \lg n! \rceil$[1]. Given a perm π in its most natural representation, i.e. the sequence $\pi(i)$, for $i = 0, \ldots, n-1$, $\pi^k(i)$ is easily computable in $k-1$ steps. Indeed, for this representation, a $\Theta(n)$ lower bound follows for computing $\pi^k(i)$ when k is large and i is on a large cycle. To facilitate the computation in constant time, one could store $\pi^k(i)$ for all i and k ($|k| \le n$, along with its cycle length), but that would require $\Theta(n^2 \lg n)$ bits. The most natural compromise is to retain $\pi^k(i)$ with $|k| \le n$ a power of 2. This $n(\lg n)^2$ bit representation easily yields a logarithmic evaluation scheme. Unfortunately we are a factor of $\lg n$ from the minimal space representation and still have a $\Theta(\lg n)$ algorithm. Our main result removes this logarithmic factor from both the time and the space terms, giving $\pi^k(i)$ in constant time and essentially minimum space. To be more specific, we demonstrate:

[1] lg denotes logarithm to the base 2

1. a representation of a perm π that takes $(1+\epsilon)n \lg n + o(n)$ bits of space, and supports $\pi()$ in $O(1)$ time and $\pi^k()$, for any k, in $O(1/\epsilon)$ time, for any $\epsilon > 0$. We also show a restricted lower bound matching this time-space trade-off.
2. a second representation of a perm π that takes $\mathcal{P}(n) + o(n)$ bits of space, and supports $\pi^k()$ for any k in $O(\lg n / \lg\lg n)$ time.

Along the way, we show that answering $\pi()$ and $\pi^{-1}()$ queries suffices to compute queries of arbitrary perm powers.

One sub-routine we develop here is a representation of a sequence of n integers from $[r]$, for some integer $r \geq 1$, that takes $n \lg r + o(n)$ bits and allows the i-th integer to be accessed in $O(1)$ time. Note that this is $\Theta(n)$ bits better than the naive representation that takes $n \lceil \lg r \rceil$ bits. As an immediate application of this result, we obtain an improvement of a similar magnitude for storing satellite information in Pagh's dictionary [13].

There are a number of motivations for succinct data structures in general, many to do with text indexing or representing huge graphs [5, 6, 11, 14]. Indeed, there has already been work on space-efficient representation of restricted classes of perms, such as the perms representing the lexicographic order of the suffixes of a string [5] or so-called approximately min-wise independent perms, used for document similarity estimation [2]. Work on succinct representation of a perm and its inverse was, for one of the authors, originally motivated by a data warehousing application. Under the indexing scheme in the system, the perm corresponding to the rows of a relation sorted under any given key was explicitly stored. It was realized that to perform certain joins, the inverse of a segment of this perm was precisely what was required. The perms in question occupied a substantial portion of the several hundred gigabytes in the indexing structure and doubling this space requirement (for the perm inverses) for the sole purpose of improving the time to compute certain joins was inappropriate. Other applications arise in Bioinformatics [1]. The more general problem of quickly computing $\pi^k()$ also has number of applications. An interesting one is determining the r^{th} root of a perm [12]. Our techniques not only solve the r^{th} power problem immediately, but can also be used to find the r^{th} root, if one exists.

The remainder of the paper is organized as follows. The next section describes some previous results on indexable dictionaries used in later sections, as well as the representation of a sequence of n integers from $[r]$, for some integer $r \geq 1$. Section 3 describes the 'shortcut' method and a matching lower bound (item (1) above) and Section 4 describes item (2) above. We assume a standard *word RAM* model with word size $\Theta(\lg n)$ bits for all our results.

2 Preliminaries

Indexable Dictionaries. Given a set $S \subseteq [m]$, an *indexable dictionary* representation [14] for S supports the following operations in constant time:

$\mathsf{rank}(x, S)$: Given $x \in [m]$, return -1 if $x \notin S$ and $|\{y \in S | y < x\}|$ otherwise;
$\mathsf{select}(i, S)$: Given $i \in [n]$, return the $i+1$-st smallest element in S.

A *fully indexable dictionary (FID)* representation supports the above rank and select operations in constant time for $\bar{S}$ (the complement of S), as well. In particular, it can also support fullrank(x, S) operation which returns $|\{y \in S | y < x\}|$ for *all* $x \in [m]$. Using the characteristic vector of S, and an auxiliary $o(m)$ bit structure to support rank and select operations in a bit vector [6, 11], it is known that:

Theorem 1. *Given a set $S \subseteq [m]$, there is a FID on S that uses $m + o(m)$ bits.*

Using some bucketing techniques and succinct encodings for prefix sums of bucket sizes, Raman, Raman and Rao [14] show the following:

Theorem 2 (Theorem 4.1 of [14]). *There is an indexable dictionary for a set $S \subseteq [m]$ of size n using at most $\lceil \lg \binom{m}{n} \rceil + o(n) + O(\lg \lg m)$ bits.*

Theorem 3 (Lemma 4.1 of [14]). *There is a FID for a set $S \subseteq [m]$ of size n using at most $\lceil \lg \binom{m}{n} \rceil + O(m \lg \lg m / \lg m)$ bits.*

Representing Numbers. We now show how to represent n numbers $a_1, \ldots, a_n$ from $[r]$ in $n \lg r + o(n)$ bits, so that we can access the i-th number $O(1)$ time. (A straightforward representation takes $n \lceil \lg r \rceil$ bits, which is $\Theta(n)$ bits more than the optimal $n \lg r$ in the worst case.)

First assume that $r \leq \lg n$. For some $z \geq 1$, we partition the input numbers into contiguous subsequences of z input numbers. We view each subsequence as an integer from $[r^z]$ and represent it using at most $\lceil z \lg r \rceil \leq z \lg r + 1$ bits. We choose z as large as possible so that $z \lg r \leq \frac{1}{2} \lg n$; this allows an individual number in a subsequence to be accessed in $O(1)$ time by looking up a precomputed table of size at most $z \cdot \lceil \lg r \rceil \cdot 2^{z \lg r + 1} = O(\sqrt{n} \lg n)$ bits. The space used is $(n/z)(z \lg r + 1) + O(\sqrt{n} \lg n) = n \lg r + O(n \lg \lg n / \lg n)$ bits, since $z = \Omega(\lg n / \lg \lg n)$. Now assume that $r > \lg n$ and let $l \geq 1$ be the smallest integer such that $k = \lfloor r/2^l \rfloor \leq \lg n - 1$. We store the sequence $\{a_i \bmod 2^l\}$ using nl bits in the obvious way. As the values a_i div 2^l are from $[k+1]$, where $k + 1 \leq \lg n$, we can store the sequence $\{a_i \text{ div } 2^l\}$ using $n \lg(k+1) + O(n \lg \lg n / \lg n)$ bits using the above method. Given i, we can easily reconstruct a_i from its "div" and "mod" values in $O(1)$ time. The space used is $n(l + \lg(k+1)) + O(n \lg \lg n / \lg n)$ bits. Since $(k+1)2^l > r \geq k2^l$, we have $\lg(k+1) + l > \lg r \geq \lg k + l$. However, $\lg(k+1) = \lg k + O(1/k)$, so $n(l + \lg(k+1)) \leq n \lg r + O(n/k)$. Since $k = \Theta(\lg n)$, the space used in this case is also $n \lg r + O(n \lg \lg n / \lg n)$ bits.

Theorem 4. *A sequence of n numbers from $[r]$ can be represented using $n \lg r + o(n)$ bits so that we can access the i-th element of the sequence in $O(1)$ time.*

From the theorem, we get a representation for an arbitrary permutation on $[n]$ taking $n \lg n + o(n)$ bits supporting $\pi()$ in constant time. To the best of our knowledge, this is the first such representation taking less than $n \lceil \lg n \rceil$ bits.

3 Near Optimal Representations

First we design a space efficient data structure that can support both π and π^{-1} in constant time. Let $t \geq 2$ be a parameter. We first represent the sequence $\pi(i)$, for $i = 0$ to $n-1$ using the representation of Theorem 4 taking $n \lg n + o(n)$ bits. Let A be this representation. In addition, we trace the cycle structure of the perm, and for every cycle whose length is at least t, we store a shortcut pointer with the elements which are at a distance of a multiple of t steps from an arbitrary starting point (this idea was used in the representation of an implicit multikey table to support logarithmic searches under any key [4]). The shortcut pointer points to the element which is t steps before it in the cycle of the perm.

More precisely, let $c_0, c_1, \ldots, c_{k-1}$ be the elements of a cycle of the perm (i.e. $\pi(c_i) = c_{i+1 \bmod k}$, for $i = 0, 1, \ldots, k-1$) where $k \geq t$. Then the indices whose π values are c_{it} for $i = 0, 1, \ldots, l = \lfloor k/t \rfloor$ are called indices with shortcut pointers, and the shortcut pointer value at c_{it} stores the index whose pi value is $c_{(i+1 \bmod l)t}$, for $i = 0, 1, \ldots, l$ (see Fig. 1). Let $s \leq n/t$ be the number of shortcut pointers after doing this for every cycle of the perm. We store the pointer values in the order of the indices with shortcut pointers (regardless of which cycle each element belongs to), using $s \lg n + O(s \lg \lg s / \lg s)$ bits using Theorem 4. Let S be this representation of the sequence of pointer values. Since $s \leq n/t$, we have used $(1 + 1/t) n \lg n + O(n \lg \lg n / \lg n)$ bits along with the representation for π.

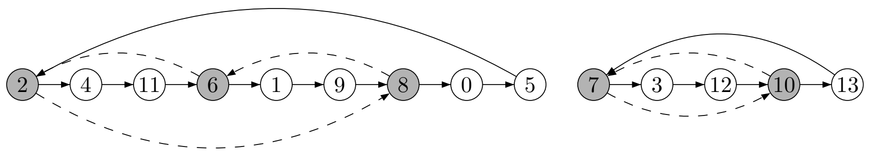


Fig. 1. Shortcut method. Solid lines denote the perm, and the dotted lines denote the back pointers. The shaded nodes indicate the positions having shortcut pointers.

We need to identify fast, indices having shortcut pointers and for those indices, their pointer values. The pointer value of an index can be found from the representation S of the sequence of pointer values, if we know the rank of the index (having a shortcut pointer) among those having shortcut pointers. The indexable dictionary of Theorem 2 can be used to represent the dictionary D of the s shortcut pointers using $\lceil \lg \binom{n}{s} \rceil + o(n)$ bits which is $O((n \lg t)/t) + o(n)$ bits as $s \leq n/t$.

The following procedure computes $\pi^{-1}(x)$ for a given x.

```
$i := x$;
while $\pi(i) \neq x$ do
if $i$ has a shortcut pointer and if its rank is $r$ (both found by querying the
    dictionary $D$)
```

```
then find the shortcut pointer value $j$ by finding the $r$-th element of the representation $S$.
else $j := \pi(i)$ (found by querying the $i$-th value of the representation $A$);
    $i := j$
endwhile
```

$\pi^{-1}(x) = i$.

Since we have a shortcut back pointer for every t elements of a cycle, the number of π computations made by the algorithm is at most $t + 1$. So the algorithm to compute π^{-1} takes at most $O(t)$ steps. Thus we have

Theorem 5. *There is a representation of an arbitrary perm π on $[n]$ using at most $(1 + 1/t)n \lg n + o(n)$ bits that can support the operations $\pi()$ in constant time, and $\pi^{-1}()$ in $O(t)$ time, for any parameter $t > 0$.*

Choosing t to be approximately $2/\epsilon$ for any positive constant $\epsilon < 1$, we have

Corollary 1. *There is a representation to store a perm π on $[n]$ using at most $(1+\epsilon)n \lg n + O(1)$ bits in which one can support $\pi()$ in $O(1)$ time and $\pi^{-1}()$ in $O(1/\epsilon)$ time, for any positive constant ϵ less than 1.*

Choosing t to be $f(n) \lg n$ for some increasing function f of n we have

Corollary 2. *There is a representation to store an arbitrary perm π on $[n]$ using at most $n \lg n + o(n)$ bits that can support $\pi()$ in constant time, and $\pi^{-1}()$ in $O(f(n) \lg n)$ time where $f(n)$ is any increasing function of n. The $o(n)$ term, here, is $O(n/f(n) + n \lg\lg n / \lg n)$.*

Optimality. Demaine and López-Ortiz [3] showed that any text index supporting linear time substring searches requires about as much space as the original text. Here, given a text T, we want to construct an index I such that given any pattern P, one can find an occurrence of P in T in $O(|P|)$ time. They consider the model in which time is counted only in terms of the number of bits probed from the text (all other computation and probes to the index and the pattern are free). They show that any index I supporting a search for a pattern P using $O(|P|)$ bit probes to the text T should have size $|I| = \Omega(|T|)$. They also show the following trade-off result:

Theorem 6 (Corollary 3.1 of [3]). *If there is an algorithm supporting substring searches of length $|P| = \lg n + o(\lg n)$ using at most $\mathcal{S} = o(\lg^2 n / \lg\lg n)$ bit probes to a text of size $|T| = n \lg n + o(n \lg n)$, then $|I| = \Omega(|T| \lg n / \mathcal{S})$.*

They show this by considering texts that are obtained by writing a random perm π (with high Kolmogorov complexity) as $T_\pi = \pi(0)\#\pi(1)\# \ldots \#\pi(n-1)$, and restrict the patterns to be $i\#$ for some $i \in [n]$. Note that searching for $i\#$ in T_π is equivalent to finding $\pi^{-1}(i)$ (i.e., $\pi^{-1}(i)$ is the position of $i\#$ in T_π). From their proof, for the RAM model with word size $\Theta(\lg n)$, one can show that

Corollary 3. *Let P be a structure that stores a perm π and answers $\pi(i)$ queries in $O(1)$ time. Then any data structure that answers $\pi^{-1}(i)$ queries using t queries to P, where t is $o(\lg n / \lg\lg n)$, requires an additional index structure taking at least $(n \lg n)/t$ bits of space.*

Proof Sketch. The t queries to the structure P can be simulated with $t(\lg n + o(\lg n))$ bit probes to the text T_π. □

This, in particular implies that the structure of Theorem 5 is 'essentially' optimal up to lower order terms.

3.1 Supporting Arbitrary Powers

There is no easier way, in the structure of Theorem 5, to compute π^k for $k > 1$ (or $k < 1$) than by repeated application of π or π^{-1}. Here we develop a succinct structure to support all powers of π (including π and π^{-1}).

Theorem 7. *Suppose there is a representation R taking $s(n)$ bits to store an arbitrary perm π on $[n]$, that supports $\pi()$ in p steps, and $\pi^{-1}()$ in q steps. Then there is a representation for an arbitrary perm on $[n]$ taking $s(n) + n + o(n)$ bits in which $\pi^k()$ for any k can be supported in time $p + q + O(1)$.*

Proof. Let π be the given perm to be represented to support all its powers. Consider its cycle representation, which is a collection of disjoint cycles of the perm (where the cycles are ordered arbitrarily). Remove the brackets and consider the resulting sequence as an array A of length n. Let $\psi(\pi)$ be the perm that maps i to the position j of i in the array; i.e j such that $A[j] = i$. Equivalently, $\psi^{-1}(j) = A[j]$. Note that $\psi(\pi)$ is not uniquely defined as it depends on the ordering of the cycles. For example, if π on 12 elements is given by (1 5 8 3)(2 4 11)(6 10)(7 0 9), then the resulting sequence is 1 5 8 3 2 4 11 6 10 7 0 9. And $\psi(\pi)$ is the perm given by $\psi(0) = 10; \psi(1) = 0, \psi(2) = 4$ and so on.

Now we will represent the perm ψ using the representation R taking $s(n)$ bits where we can support $\psi(i)$ and $\psi^{-1}(i)$ in time p and q respectively. In addition, we need to store the starting points (or the lengths) of each cycle of π efficiently. Let F be the indices of the starting points of the cycles of π. We store F using the FID representation of Theorem 1 taking $n + o(n)$ bits. This justifies the space usage in the theorem, and we are ready to explain how powers of π can be computed. To compute $\pi^k(i)$, we first find $j = \psi(i)$. Next we need to find the cycle C that contains i, and its length. Querying $\mathsf{fullrank}(j, F) = p$ gives the number of elements of F less than j which gives the cycle number (in the left to right order of the cycles) of the cycle C. Then the length l of the cycle C_j is $\mathsf{select}(p + 1, F) - \mathsf{select}(p, F)$. Let $r = \mathsf{select}(p, F)$ be the index where the p-th cycle starts. We find $s = r + ((j - r + k) \bmod l)$ and return $\psi^{-1}(s)$ which gives $\pi^k(i)$. Note that this works for both $k > 0$ and $k < 0$. Since the FID representation supports select and fullrank operation in constant time, we have the theorem. □

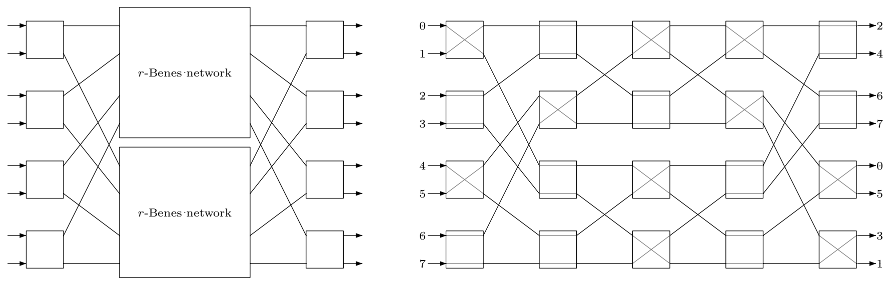
As an immediate corollary, we get from Corollary 1

Corollary 4. *There is a data structure to represent any perm* π *on* $[n]$ *using* $(1+\epsilon)n\lg n + O(1)$ *bits in which we can support the operation* $\pi^k(i)$ *for any* k *in constant time, for any positive constant* ϵ *less than* 1.

4 Optimal-space Representation

The Benes network. Our results in this section are based on the Benes network, which is a communication network composed of a number of switches, and which we now outline (see [10] for details). Each switch has 2 inputs x_0 and x_1 and 2 outputs y_0 and y_1 and can be configured either so that x_0 is connected to y_0 (i.e. a packet that is input along x_0 comes out of y_0) and x_1 is connected to y_1, or the other way around. An r-Benes network has 2^r inputs and outputs, and is defined as follows. For $r = 1$, the Benes network is a single switch with 2 inputs and 2 outputs. An $(r+1)$-Benes network is composed of 2^{r+1} switches and two r-Benes networks, connected as as shown in Fig. 2(a). A particular setting of the switches of a Benes network *realises* a perm π if a packet introduced at input i comes out at output $\pi(i)$, for all i (Fig. 2(b)). The following properties are either easy to verify or well-known [10].

- An r-Benes network has $r2^r - 2^{r-1}$ switches, and every path from an input to an output passes through $2r-1$ switches;
- For every perm π on $[2^r]$ there is a setting of the switches that realises π.



(a) construction of $(r+1)$-Benes network

(b) Benes network realising the permutation (4 7 0 6 1 4 2 3)

Fig. 2. The Benes network (construction) and an example

The restriction that the number of inputs be a power of 2 will prove to be a severe one in our context. We now define a family of Benes-like networks that admit greater flexibility in the number of inputs, namely the (q,r)-Benes networks, for integers $r \geq 0, q > 0$. First, we define a *q-permuter* to be a communication network that has q inputs and q outputs, and realises any of the $q!$ perms of its inputs by some settings of its switches (an r-Benes network is a 2^r-permuter).

Taking $p = q2^r$, a (q,r)-modified Benes network is a q-permuter for $r = 0$, and for $r > 0$ it is composed of p switches and two $(q, r-1)$-Benes networks, connected together in exactly the same way as a standard Benes network.

Lemma 1. *Let $q > 0, r \geq 0$ be integers and take $p = q2^r$. Then:*

1. *A (q,r)-Benes network consists of $qr2^r$ switches and 2^r q-permuters;*
2. *For every perm π on $[p]$ there is a setting of the switches of the (q,r)-Benes network that realises π.*

Proof. (1) is obvious; (2) can be proved in the same way as for a standard Benes network. □

Representing Perms. Clearly, Benes networks may be used to represent perms. For example, if $n = 2^r$, a representation of a perm π on $[n]$ may be obtained by configuring an r-Benes network to realize π and then listing the settings of the switches in some canonical order (e.g. level-order). This represents π using $r2^r - 2^{r-1} = n \lg n - n/2$ bits. Given i, one can trace the path taken by a packet at input i by inspecting the appropriate bits in this representation, and thereby calculate $\pi(i)$ in $O(\lg n)$ time[2]. In fact, by tracing the path back from output i we can also compute $\pi^{-1}(i)$ in $O(\lg n)$ time. To summarise:

Proposition 1. *When $n = 2^r$ for some integer $r > 0$, there is a representation of an arbitrary perm π on $[n]$ that uses $n \lg n - n/2$ bits and can support the operations $\pi()$ and $\pi^{-1}()$ in $O(\lg n)$ time.*

We now consider representations based on (q,r)-Benes networks; these will replace the central q-permuters with alternative representations of perms.

Proposition 2. *If $q \leq \lg n/(2 \lg \lg n)$, there is a representation of an arbitrary perm π on $[q]$ that supports $\pi()$ and $\pi^{-1}()$ in $O(1)$ time. This assumes access to a pre-computed table of size $O(\sqrt{n} \lg n)$ bits that does not depend upon π.*

Proof. We represent π implicitly, e.g. as the index of π in a canonical enumeration of all perms on $[q]$. The calculation of $\pi()$ (or $\pi^{-1}()$) is done by table lookup; the size of the required table is easily seen to be $O(\sqrt{n} \lg n)$ bits. □

Using the representation of Proposition 2, we now obtain:

Lemma 2. *If $p = q2^r$ for integers $\lg n/(4 \lg \lg n) < q \leq \lg n/(2 \lg \lg n)$ and $r \geq 0$, then there is a representation of an arbitrary perm π on $[p]$ that uses $\mathcal{P}(p) + \Theta((p \lg p)/q)$ bits, and supports $\pi()$ and $\pi^{-1}()$ in $O(r)$ time each. This assumes access to a pre-computed table of size $O(\sqrt{n} \lg n)$ bits that does not depend upon π.*

[2] Indeed, in $O(\lg n)$ bit-probes.

Proof. Consider a (q, r)-Benes network that realises π; we list all the switch settings of the outer $2r$ layers of switches as in Proposition 1. For each of the q-permuters we represent the perm realised by it using Proposition 2. Computing $\pi()$ or π^{-1} involves the inspection of $2r$ bits in the outer layers, plus a table lookup in the centre. We now calculate the space used. Note that:

$$\begin{aligned}\mathcal{P}(p) = p\lg(p/e) + \Theta(\lg p) &= q2^r(r + \lg(q/e)) + \Theta(\lg p)\\ &= qr2^r + 2^r(q\lg(q/e)) + \Theta(\lg p)\end{aligned}$$

By Lemma 1, space used by the above representation (excluding lookup tables) is $qr2^r + 2^r\mathcal{P}(q) = qr2^r + 2^r q\lg(q/e) + \Theta(2^r \lg pq) = \mathcal{P}(p) + \Theta((p\lg p)/q)$. □

For perms on arbitrary $[n]$, we need the following proposition:

Proposition 3. *For all integers $p, t \geq 0$, $p \geq t$ there is an integer $p' \geq p$ such that $p' = q2^r$ for integers $t < q \leq 2t$ and $r \geq 0$, and $p' < p(1 + 1/t)$.*

Proof. Take q to be $\lceil p/2^r \rceil$, where r is the power of 2 that satisfies $t < p/2^r \leq 2t$. Note that $p' < (p/2^r + 1) \cdot 2^r = p(1 + 2^r/p) < p(1 + 1/t)$. □

Theorem 8. *An arbitrary perm π on $[n]$ may be represented using $\mathcal{P}(n) + o(n)$ bits, such that $\pi()$ and $\pi^{-1}()$ can both be computed in $O(\lg n/\lg\lg n)$ time.*

Proof. Let $t = (\lg n)^2$. We first consider representing a perm ψ on $[l]$ for some integer l, $t < l \leq 2t$. To do this, we find an integer $p = l(1 + O(\lg\lg n/\lg n))$ that satisfies the preconditions of Lemma 2; such a p exists by Proposition 3. An elementary calculation shows that $\mathcal{P}(p) = \mathcal{P}(l)(1 + O(\lg\lg n/\lg n)) = \mathcal{P}(l) + O(\lg n(\lg\lg n)^2)$. We extend ψ to a perm on $[p]$ by setting $\psi(i) = i$ for all $l \leq i < p$ and represent ψ. By Lemma 2, ψ can be represented using $\mathcal{P}(p) + \Theta(\lg n(\lg\lg n)^2) = \mathcal{P}(l) + \Theta(\lg n(\lg\lg n)^2)$ bits such that $\psi()$ and $\psi^{-1}()$ operations are supported in $O(\lg\lg n)$ time, assuming access to a pre-computed table of size $O(\sqrt{n}\lg n)$ bits.

Now we represent π as follows. We choose an $n' \geq n$ such that $n' = n(1 + 1/(\lg n)^2)$ and $n' = q2^r$ for some integers q, r such that $t < q \leq 2t$. Again we extend π to a perm on $[n']$ and represent this extended perm. As in Lemma 2 we start with a (q, r)-Benes network that realises π and write down the switch settings of the $2r$ outer levels in level-order. The perms realised by the central q-permuters are represented as above. Ignoring any pre-computed tables, the space requirement is $qr2^r + 2^r(\mathcal{P}(q) + \Theta(\lg n(\lg\lg n)^2))$ bits, which is again easily shown to be $\mathcal{P}(n') + \Theta((n'\lg n')/q + 2^r\lg n(\lg\lg n)^2)) = \mathcal{P}(n') + \Theta(n(\lg\lg n)^2/(\lg n))$ bits. Finally, as above, $\mathcal{P}(n') = (1 + O(1/(\lg n)^2))\mathcal{P}(n)$, but the space requirement is still $\mathcal{P}(n) + \Theta(n(\lg\lg n)^2/(\lg n)) = \mathcal{P}(n) + o(n)$ bits.

The running time for $\pi()$ and $\pi^{-1}()$ is clearly $O(\lg n)$. To improve this to $O(\lg n/\lg\lg n)$, we now explain how to step through multiple levels of a Benes network in $O(1)$ time, taking care not to increase the space consumption significantly. Consider a (q, r)-Benes network and let $t = \lfloor \lg\lg n - \lg\lg\lg n \rfloor - 1$. Consider the case when $t \leq r$ (the other case is easier), and consider input number 0 to the (q, r)-Benes network. Depending upon the settings of the switches, a packet entering at input 0 may reach any of 2^t switches in t steps A little thought shows that the only packets that could appear at the inputs to these 2^t switches are the 2^{t+1} packets that enter at inputs $0, 1, k, k+1, 2k, 2k+1, \ldots$, where $k = q2^{r-t}$. The settings of the $t2^t$ switches that could be seen by any one of these packets suffice to determine the next t steps of *all* of these packets. Hence, when writing down the settings of the switches of the Benes network in the representation of π, we write all the settings of these switches in $t2^t \le (\lg n)/2$ consecutive locations. Using table lookup, we can then step through t of the outer $2r$ layers of the (q, r)-Benes network in $O(1)$ time. Since computing the effect of the central q-permuter takes $O(\lg\lg n)$ time, we see that the overall running time is $O(r/t + \lg\lg n) = O(\lg n/\lg\lg n)$. □

Corollary 5. *Let f be a bijection from $S \subseteq [l]$ to $T \subseteq [m]$, and let $r = |S| = |T|$. We can represent this bijection so that we can compute $f(x)$ and $f^{-1}(y)$ for all $x \in S$ and $y \in T$, respectively, in $O(\lg r/\lg\lg r)$ time. The representation takes $\left\lceil \lg \binom{l}{r} \right\rceil + \left\lceil \lg \binom{m}{r} \right\rceil + \mathcal{P}(r) + o(r) + O(\lg\lg l + \lg\lg m)$ bits.*

Proof. As explained in the introduction, we represent S and T as a pair of indexable dictionaries, with a "connecting" perm π on $[r]$. The space and time upper bounds follow from Theorems 2 and 8. □

4.1 Powers of π

Using Theorems 7 and Theorem 8, one can get a structure that supports arbitrary powers of π in $O(\lg n/\lg\lg n)$ time using $\mathcal{P}(n) + n + o(n)$ bits of space. We show how to reduce the space to optimal $\mathcal{P}(n) + o(n)$ bits while retaining the query time bounds. To acheive this bound, we need to store the cycle starting points using $o(n)$ bits in the representation of Theorem 7. Note that the algorithm to compute $\pi^k()$ actually requires $\mathsf{fullrank}$ operation and so just an indexable dictionary doesn't suffice. Using FID of Theorem 3 to represent F will result in the space (in addition to $s(n)$) of $\left\lceil \lg \binom{n}{|F|} \right\rceil + o(n)$ bits (since $m = n$ here). For an arbitary perm, the number of cycles, and hence $|F|$ could be $\Omega(n)$ and so this bound could also be $O(n)$.

We develop below a different structure that takes $o(n)$ bits. We first order the cycles in non decreasing order of their lengths. Then we distinguish between *long* cycles whose length is greater than $\lceil \lg^2 n \rceil$ and *short* cycles whose length is at most $\lceil \lg^2 n \rceil$, and represent their starting points differently.

We take the representation of the starting points of long cycles first. Let S be the set of all starting points of long cycles. Let $|S| = k \le n/\lg^2 n$. For this range of k, Theorem 3 gives an $o(n)$ bit FID structure for S, but we develop a simpler structure here.

To support $\mathsf{select}(i, S)$ operation we simply store the elements in sorted order using $k \lceil \lg n \rceil$ bits which is $o(n)$. To support $\mathsf{rank}(j, S)$ operation on S, we first divide the universe $[n]$ into blocks of size $\lceil \lg n \rceil$. Then we keep the set E of indices (from 1 to $n/\lceil \lg n \rceil$) having non-empty blocks in an FID of Theorem

1 using $n/\lceil \lg n \rceil + o(n)$ bits. I.e. we have a bit vector for each block indicating whether or not it is non-empty and keep an auxiliary structure for this bit vector E (of size $n/\lceil \lg n \rceil$) to support the rank operation on E. Then we represent the non-empty blocks completely using a bit vector F of at most $k\lceil \lg n \rceil$ (since at most k of the blocks can be non-empty) and build an auxiliary structure for this bit vector (of size at most $n/\lg n$) to support rank and select operations on F.

Now to find $\mathsf{fullrank}(i, S)$, we find the number of non-empty blocks up to the block containing i by querying $r = \mathsf{fullrank}(i/\lceil \lg n \rceil, E)$. If the block containing i is non-empty, this gives the rank of the block among the non-empty blocks and the position of i in the bit vector F. So querying rank up to that position in F gives the $\mathsf{fullrank}(i, S)$. If the block containing i is empty, then $s = \mathsf{select}(r, E)$ gives the position of the previous non-empty block. Then $\mathsf{fullrank}(s\lceil \lg n \rceil, F)$ gives the answer to $\mathsf{fullrank}(i, S)$.

To represent the starting points of the short cycles, we first construct the multiset M that contains, for every $i = 1$ to $\lceil \lg^2 n \rceil$, $m_i = \sum_{j \le i} j * n_j$ where n_j is the number of cycles of length j. This is a multiset since if there is no cycle of length $i+1$, then $n_{i+1} = 0$ and hence $m_i = m_{i+1}$. For example, suppose $\lceil \lg^2 n \rceil = 8$ and in π there are 5 cycles of length 1, 4 cycles of length 4, 5 cycles of length 5 and 6 cycles of length 8, and 0 cycles of remaining lengths (up to 8). Then the multiset M we need to store is $\{5, 5, 5, 21, 46, 46, 46, 94\}$.

Let D be the set of distinct elements of M, and let R be the sequence of multiplicities of elements of D in increasing order. I.e., the i-th element of R is the number of occurrences of the i-th smallest element of D. Let P be the sequence of partial sums of elements of R. I.e., the i-th element of P is the sum of the first i elements of R. For the example outlined above, $D = \{5, 21, 46, 94\}$ and $P = \{3, 4, 7, 8\}$. We represent D and P using the set representation outlined earlier (to represent S), that can support $\mathsf{fullrank}()$ and $\mathsf{select}()$ operations in constant time. We will also explicitly store the last element L of M (which gives the starting point of the first long cycle). Note that $|D| = |P| \le \lceil \lg^2 n \rceil$ and so the representation for D and P (and hence M) takes $O(\lg^3 n)$ bits. Hence along with the space for representing S ($O(n/\lg n)$ bits) the space used is $o(n)$.

From the proof of Theorem 7, we see that, to compute $\pi^k(i)$, we need to find the following two quantities: l, the length of the cycle containing i and r, the starting position of the cycle containing i. If i is in a long cycle (which can be found by comparing the position $j = \psi(i)$ of i with L), then the $\mathsf{fullrank}()$ and $\mathsf{select}()$ operations on S gives these information as in the proof of Theorem 7. We should just remember to add L to the starting point of these cycles.

If i falls in a short cycle, we first find $d = \mathsf{fullrank}(j, D)$ (note j is the position of i in the list). This gives the number of distinct elements in M less than j. Then $s = \mathsf{select}(d, P)$ gives the total number of elements less than j in M. So $l = s + 1$ is the length of the cycle containing i. If $\mathsf{select}(d, D) = t$, then $t + 1$ is the starting point of the groups of cycles of length l in π. So $(j - t) \bmod l$ gives the position of i in its cycle, and so $r = j - ((j - t) \bmod l) + 1$ is the starting point r of its cycle. With these operations supported in constant time, we have:

Theorem 9. *Suppose there is a representation R taking $s(n)$ bits to store an arbitrary perm π on $[n]$, that supports $\pi()$ in p steps, and $\pi^{-1}()$ in q steps. Then there is a representation for an arbitrary perm π on $[n]$ taking $s(n) + o(n)$ bits in which $\pi^k()$ for any k can be supported in time $p + q + O(1)$.*

As an immediate corollary, we get, from Theorem 8

Corollary 6. *There is a representation to store an arbitrary perm π on $[n]$ using at most $\lceil \lg n! \rceil + o(n)$ bits that can support $\pi^k()$ for any k in $O(\lg n / \lg \lg n)$ time.*

Acknowledgements The authors thank Rasmus Pagh for directing attention to the problem of representing a sequence of numbers and Rick Thomas and Eugene Zima for several useful discussions.

References

1. D. A. Bader, M. Yan, B. M. W. Moret. A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. University of New Mexico Technical Report HPCERC2001-005 (August 2001): http://www.hpcerc.unm.edu/Research/tr/HPCERC2001-005.pdf
2. A. Z. Broder, M. Charikar, A. M. Frieze and M. Mitzenmacher. Min-wise independent permutations. *J. Computer System Sci.* **60** 630–659 (2000).
3. E. D. Demaine and A. López-Ortiz. A linear lower bound on index size for text retrieval. *Journal of Algorithms*, to appear, 2003.
4. A. Fiat, J. I. Munro, M. Naor, A. A. Schäffer, J.P. Schmidt and A. Siegel. An implicit data structure for searching a multikey table in logarithmic time. *Journal of Computer and System Sciences*, **43** 406–424 (1991).
5. R. Grossi and J. S. Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. In *Proceedings of the ACM Symposium on Theory of Computing*, 397–406, 2000.
6. G. Jacobson. Space-efficient static trees and graphs. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, 549–554, 1989.
7. D. E. Knuth. Efficient representation of perm groups. *Combinatorica* **11** 33–43 (1991).
8. D. E. Knuth. *The Art of Computer Programming, vol. 1: Fundamental Algorithms.* Computer Science and Information Processing. Addison-Wesley, 1973.
9. D. E. Knuth. *The Art of Computer Programming, vol. 3: Sorting and Searching.* Computer Science and Information Processing. Addison-Wesley, 1973.
10. F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees and Hypercubes.* Computer Science and Information Processing. Morgan Kauffman, 1992.
11. J. I. Munro and V. Raman. Succinct representation of balanced parentheses and static trees. *SIAM Journal on Computing*, **31** (3):762-776, 2002.
12. N.Pouyanne. On the number of permutations admitting an m-th root. *The Electronic Journal of Combinatorics*, 9 (2002), #R3.
13. R. Pagh. Low redundancy in static dictionaries with constant query time. *SIAM Journal on Computing*, **31** (2):353–363, 2001.
14. R. Raman, V. Raman and S. S. Rao. Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 233–242, 2002.