Efficient algorithms on sets of permutations, dominance, and real-weighted APSP

Raphael Yuster [*]

Abstract

Sets of permutations play an important role in the design of some efficient algorithms. In this paper we design two algorithms that manipulate sets of permutations. Both algorithms, each solving a different problem, use fast matrix multiplication techniques to achieve a significant improvement in the running time over the naive solutions.

For a set of permutations $P \subset S_n$ we say that i k-dominates j if the number of permutations $\pi \in P$ for which $\pi(i) < \pi(j)$ is k. The *dominance matrix* of P is the $n \times n$ matrix D_P where $D_P(i,j) = k$ if and only if i k-dominates j. We give an efficient algorithm for computing D_P using fast *rectangular* matrix multiplication. In particular, when $|P| = n$ our algorithm runs in $O(n^{2.684})$ time. Computing the dominance matrix of permutations is computationally equivalent to the dominance problem in computational geometry. Thus, our algorithm slightly improves upon a well-known $O(n^{2.688})$ time algorithm of Matousek for the dominance problem. Permutation dominance is used, together with several other ingredients, to obtain a truly sub-cubic algorithm for the *All Pairs Shortest Paths* (APSP) problem in real-weighted directed graphs, where the number of distinct weights emanating from each vertex is $O(n^{0.338})$. A special case of this algorithm implies an $O(n^{2.842})$ time algorithm for real vertex-weighted APSP, which slightly improves a recent result of Chan [STOC-07].

A set of permutations $P \subset S_n$ is *fully expanding* if the product of any two elements of P yields a distinct permutation. Stated otherwise, $|P^2| = |P|^2$ where $P^2 \subset S_n$ is the set of products of two elements of P. We present a randomized algorithm that computes $|P^2|$ and hence decides if P is fully expanding. The algorithm also produces a table that, for any $\sigma_1, \sigma_2, \sigma_3, \sigma_4 \in P$, answers the query $\sigma_1\sigma_2 = \sigma_3\sigma_4$ in $\tilde{O}(1)$ time. The algorithm uses, among other ingredients, a combination of fast matrix multiplication and polynomial identity testing. In particular, for $|P| = n$ our algorithm runs in $O(n^\omega)$ time where $\omega < 2.376$ is the matrix multiplication

exponent. We note that the naive deterministic solution for this problem requires $\Theta(n^3)$ time.

1 Introduction

Sets of permutations play an important role in the design of some efficient algorithms and their derandomization (see, e.g., [1, 3]). In this paper we consider several problems involving computational aspects of permutation sets, and some applications.

In order to describe our results which use fast matrix multiplication as an important ingredient, we need a definition. Let $M(a,b,c)$ be the time needed to multiply an $a \times b$ matrix by a $b \times c$ matrix over an arbitrary ring R. Let $\omega(r,s,t)$ be the minimal exponent for which $M(n^r, n^s, n^t) = O(n^{\omega(r,s,t)})$. (by "time" we assume that each matrix element can be represented with $O(\log n)$ bits). A seminal result of Coppersmith and Winograd shows that $\omega = \omega(1,1,1) < 2.376$ [7].

An important computational aspect of permutation sets is that of *dominance*. For a set of permutations $P \subset S_n$ we say that i k-dominates j if the number of permutations $\pi \in P$ for which $\pi(i) < \pi(j)$ is k. The *dominance matrix* of P is the $n \times n$ matrix D_P where $D_P(i,j) = k$ if and only if i k-dominates j. Computing dominance matrices is a key (sometimes implicit) ingredient of several well known algorithms. Fredman's non-uniform strategy for computing distance products in sub-cubic time implicitly uses permutation dominance [8]. Even more directly, Matousek's well-known result on computing dominances of vector sets in R^n [11] is computationally equivalent to permutation dominance (see Section 2). In the vector dominance problem we are given a set of vectors $V \subset R^n$, and we wish to compute, for any two vectors $u, v \in V$, whether u dominates v, that is, whether $u_i \leq v_i$ for all coordinates $i = 1, \ldots, n$. More informatively, we wish to compute $D(u,v)$, the number of coordinates i for which $u_i \leq v_i$. Matousek's algorithm solves this problem in $O(n^{(\omega+3)/2})$ time, if $|V| = n$. Permutation dominance is an important ingredient in more recent results; on all-pairs shortest paths [4], on all-pairs bottleneck paths [14], and on finding heaviest vertex-weighted subgraphs [13]. In Section 2 we show how to

compute the dominance matrix D_P slightly faster than was previously known. In particular, when $|P| = n$ our algorithm runs in $O(n^{2.684})$ time. This is slightly faster than the $O(n^{2.688})$ time which follows from Matousek's result.

Permutation dominance is used as an important ingredient in a new algorithm for the well-known All-Pairs Shortest Paths (APSP) problem in real weighted directed graphs, which is presented in Section 3. Given a weighted directed graph $G = (V, E, w)$ where $w : E \to R$, the *distance matrix D* is the $V \times V$ matrix with $D(u, v) = \delta(u, v)$, the shortest distance from u to v. The goal in the APSP problem is to compute D, and to compute a concise data structure representing the shortest paths. No truly sub-cubic algorithm for the APSP problem is known. The presently fastest algorithm for the APSP problem, due to Chan, runs in $O(n^3 \log^3 \log n / \log^2 n)$ time [5] (as usual, computations on real numbers assume the addition-comparison model). Sub-cubic algorithms are known in the case where the weights are integers that are not too large. Zwick [16] obtained an algorithm for APSP that runs in $O(n^{2.575})$ time if the (absolute value of the) weights are bounded integers. His algorithm is sub-cubic if the largest absolute integer weight is $o(n^{3-\omega})$.

We present an APSP algorithm for the real weighted case that is truly sub-cubic as long as the set of possible weights emanating from each vertex is not too large, but still *much larger* than a constant. More precisely, for $v \in V$, let $W(v)$ be the set of distinct weights of edges emanating from v and let $k(G) = \max_{v \in V} |W(v)|$. Trivially, $k(G) \leq n - 1$ for any graph. Our algorithm shows that APSP can be solved in truly sub-cubic time as long as $k(G) \leq n^{0.338}$. Notice that this allows the possibility of up to $\Theta(n^{1.338})$ distinct weights in the graph. This follows from the the following result, which considers a more general setting. Let μ be the solution to $\omega(1 + \mu, 1, 1 + \mu) = 3$. It is known (see Section 3) that $\mu > 0.338$.

THEOREM 1.1. *Let $G = (V, E, w)$ be a real weighted directed graph with $|V| = n$ and with $k(G) = \Theta(n^\rho)$, where $\rho < \mu$. There is an algorithm that solves the APSP problem in $\tilde{O}(n^{3 - \gamma/2 + \rho/2})$ time, where γ is the solution to the equation $\omega(1 + \rho, 1, 1 + \gamma) = 3 + \rho - \gamma$.*

An important consequence of Theorem 1.1 occurs is the special case $\rho = 0$ (the case where $k(G)$ is bounded by a constant). The solution to $\omega(1, 1, 1 + \gamma) = 3 - \gamma$ satisfies $\gamma > 0.3165$ (see Lemma 2.1). Thus, in this case the APSP problem can be solved in $O(n^{2.842})$ time. This case also applies to *vertex-weighted APSP* by the following simple reduction: Given a real vertex-weighted graph, construct edge weights by assigning

$w(u, v) = w(u)$. Clearly, a shortest path from u to v of weight $\delta(u, v)$ in the edge-weighted graph corresponds to a shortest path of weight $\delta(u, v) + w(v)$ in the vertex-weighted graph. Furthermore, $k(G) = 1$ in the edge-weighted graph. The $O(n^{2.842})$ running time we obtain slightly improves upon the $O(n^{2.844})$ time for vertex-weighted APSP obtained by Chan [5]. Theorem 1.1 can be used to obtain a sub-cubic algorithm for *approximate* shortest paths in a general APSP instance (namely, when there is no restriction on the number of distinct weights emanating from each vertex) by scaling and rounding the weights. For example, if the largest absolute weight in the graph is bounded, linear scaling into $O(n^{0.338})$ weights achieves an additive approximation of $O(n^{0.662})$. If the smallest absolute weight is 1 and the largest is W, then logarithmic scaling into $O(n^{0.338})$ weights achieves a stretch (multiplicative approximation) of $\epsilon = n^{-0.338} \log W$ in sub-cubic time. Zwick's scaling algorithm [16] achieves a stretch of $\epsilon = n^{-0.624} \log W$ in sub-cubic time, but only for instances that do not contain negative weights.

We now turn to our next result on permutation sets. As a subset of an algebraic group, a carefully selected relatively small set of permutations can be used to generate much larger sets. This is important, for example, if resources such as memory are limited. More formally, if P is a subset of a group, we denote by P^k the subset of the same group that is generated by all possible products of k elements of P. Notice that $|P^k| \leq |P|^k$, and if $|P^k| = \eta |P|^k$ we say that P is (η, k)-expanding, or just *fully k-expanding* if $\eta = 1$. For example, the set $P = \{(4231), (3124)\} \subset S_4$ is fully 2-expanding as P^2 contains four distinct elements. However, $P = \{(4231), (3124), (1243)\} \subset S_4$ is not fully 2-expanding. In this case $|P^2| = 7 < 9$. If P is fully k-expanding, then we can have an application store only $|P|$ elements (the generating elements), yet still have reference to a much larger set of elements that are *guaranteed* to be distinct, via group element product.

A set of permutations $P \subset S_n$ of $\{1, \ldots, n\}$ is *k-free* if the identity permutation cannot be obtained as a product of k (not necessarily distinct) elements of P. Clearly, P is 1-free if and only if $id \notin P$ and P is 2-free if and only if $\pi \in P$ implies $\pi^{-1} \notin P$. It is easy to decide if P is fully k-expanding in $O(p^k n)$ time where $|P| = p$, using the naive algorithm (we assume that k is a constant). Similarly, one can decide if P is k-free in $O(p^{\lceil k/2 \rceil} n)$ time, as follows. Let P^{-1} be the inverse of all elements of P. Clearly, P^{-1} can be constructed in $O(pn)$ time. Since P is k-free if and only if $P^{\lceil k/2 \rceil} \cap P^{-\lfloor k/2 \rfloor} = \emptyset$, the algorithm follows (to check the intersection emptiness we can sort the elements in both sets using radix sort).

Already for $k = 2$, and with, say, $p = n$, the naive algorithm for checking fully 2-expansion, as well as computing $|P^2|$, requires $\Theta(n^3)$ time, and, similarly, checking 3-freeness requires $\Theta(n^3)$ time. Thus, an obvious question is whether we can do better. This task seems to be quite difficult. Any sub-cubic algorithm for 3-freeness, or 2-expansion, must circumvent the construction of P^2, since the size of P^2 has a worst case (and also average case) lower bound of $\Theta(n^3)$. Furthermore, since P is an arbitrary set of, say, n permutations, there does not seem to be any algebraic structure that may shortcut our way to the answer. The proof of our first result shows that, at the price of randomization, one can impose an *efficiently computable* algebraic structure and obtain a significantly faster algorithm.

THEOREM 1.2. *Fix a positive integer $k \geq 3$. Let $P \subset S_n$ with $|P| = n^\beta$. There is an algorithm that decides, whp, if P is k-free. If P is not k-free, then a sequence $\pi_1, \ldots, \pi_k$ so that $\pi_1\pi_2\cdots\pi_k = id$ is obtained. The running time is $O(n^\alpha)$ where $\alpha = \omega(\beta\lceil(k-2)/4\rceil, 1, \beta\lceil k/4\rceil)$. In particular, if $|P| = n$ and $k \in \{3, 4\}$ the algorithm runs in $O(n^\omega)$ time.*

The same algorithm can be used to compute $|P^k|$ and, in particular, detect fully k-expansion. The running time in this case is $O(n^\alpha)$ where $\alpha = \omega(\beta\lceil k/2\rceil, 1, \beta\lfloor k/2\rfloor)$. Thus, fully 2-expansion of an n-element set can be decided in $O(n^\omega)$ time. It should be noted that the algorithm is of a Las Vegas type. It never yields an erroneous answer. With very small probability the algorithm would report failure. Alternatively, by repeatedly running the algorithm one obtains an algorithm that always gives the correct result in the *expected* stated running time. The proof of Theorem 1.2 appears in Section 4. The final section contains some concluding remarks and open problems.

2   Dominance matrix via rectangular matrix multiplication

In this section we show how to compute permutation dominance using rectangular matrix multiplication. For simplicity, we state the result for the case $|P| = n$. The generalization to other sizes of P will be obvious.

Let us first show the simple computational equivalence between permutation dominance and the vector dominance problem. Recall that in the vector dominance problem we have n vectors in R^n, denoted $v_1, \ldots, v_n$, and we need to compute an $n \times n$ matrix D so that $D(i, j) = |\{k \mid v_{ik} \leq v_{jk}\}|$. If $P = \{\pi_1, \ldots, \pi_n\} \subset S_n$ we define the vector v_i by $v_{ij} = \pi_j(i)$. Clearly, a solution to the dominance problem for the vectors $v_1, \ldots, v_n$ implies the permutation dominance matrix.

For the other direction, one needs to be a bit more careful, since two vectors may be equal in certain coordinates. Let $D^+(i, j) = |\{k \mid v_{ik} < v_{jk}$ or $i < j$ and $v_{ik} = v_{jk}\}|$. Let $D^-(i, j) = |\{k \mid v_{ik} < v_{jk}$ or $j < i$ and $v_{ik} = v_{jk}\}|$. Clearly, $D(i, j) = D^+(i, j)$ if $i < j$ and $D(i, j) = D^-(i, j)$ if $i > j$. Computing $D^+(i, j)$ (and, equivalently, computing $D^-(i, j)$) can be reduced to computing permutation dominance. Define a permutation π_k so that $\pi_k(i) < \pi_k(j)$ if and only if $v_{ik} < v_{jk}$ or $v_{ik} = v_{jk}$ and $i < j$. Notice that π_k can easily be computed in $O(n \log n)$ by sorting the numbers in the k'th coordinate of the vectors, and breaking ties lexicographically. Thus, the set of n permutations $P = \{\pi_1, \ldots, \pi_n\}$ can be computed in $O(n^2 \log n)$ time. (Notice that the computational equivalence that we claim is up to a logarithmic factor if permutation dominance can be solved in $o(n^2 \log n)$ time, an unrealistic assumption. Otherwise, it is exact computational equivalence). Now, clearly, D_P is precisely $D^+(i, j)$.

We require the following bound of Huang and Pan [10] on rectangular matrix multiplication.

LEMMA 2.1. *Let $r > 1$ be fixed. For any fixed positive integer q and real $\beta \in \{0.005, 0.05\}$,*

$$\omega(1, r, 1) \leq \frac{\log Z}{(1 - \beta)\log q}$$

where Z is

$$\frac{\beta^\beta((1+r)(1-\beta))^{(1+r)(1-\beta)}(1+r\beta)^{1+r\beta}(q+2)^{2+r}}{(2+r)^{2+r}} .$$

Huang and Pan used their result to obtain, in particular, that $\omega(1, 2, 1) < 3.334$ (using $r = 2$, $q = 9$, and $\beta = 0.016$ in Lemma 2.1), which was a breakthrough, as the previous best bound prior to their result was the obvious $\omega + 1 < 3.376$. We shall require a different r when applying Lemma 2.1. For the rest of this section we fix a parameter $r = 1.3165$ and let $s = r - 1 = 0.3165$.

PROPOSITION 2.1. *Let $P \subset S_n$ be a set of n permutations. Then D_P can be computed in $O(n^{2.684})$ time.*

Proof: We describe a modified version of Matousek's idea [11]. In the following description we assume that n^s and n^{1-s} are integers; this can obviously be assumed since the result is asymptotic. The noted difference between the following description and Matousek's original proof is that we actually compute the sum of several matrix products "at once" instead of computing each matrix product separately, and then summing up the results, as done in the original proof (we observe that we actually do not need the intermediate resulting products; only their sum). This enables us to use rectangular matrix multiplication.

Let $K = \{1, \ldots, n^s\}$. We define two $n \times n^r$ matrices A and B as follows. The rows of both matrices are indexed by the index set $[n]$ and the columns are indexed by $K \times [n]$. We set $A(i, (k, j)) = 1$ if $\pi_j(i) \in [(k-1)n^{1-s}+1, \ldots, kn^{1-s}]$. Otherwise, $A(i, (k, j)) = 0$. We set $B(i, (k, j)) = 1$ if $\pi_j(i) > kn^{1-s}$. Otherwise, $B(i, (k, j)) = 0$. Next, we compute $C = AB^T$ in $O(n^{\omega(1,r,1)})$ time. Notice that $C(i, j)$ is precisely the number of t so that $\lceil \pi_t(i)/n^{1-s} \rceil < \lceil \pi_t(j)/n^{1-s} \rceil$.

Next, we compute an $n \times n$ matrix E so that $E(i, j)$ is precisely the number t so that $\pi_t(i) < \pi_t(j)$ and $\lceil \pi_t(i)/n^{1-s} \rceil = \lceil \pi_t(j)/n^{1-s} \rceil$. Obviously, $E + C = D_P$. Computing E is done precisely as in Matousek's proof. We first initialize all n^2 entries of E to zero. For each pair (i, t), suppose $\pi_t(i) = \ell$. Set $b = \lceil \ell/n^{1-s} \rceil$. For each j so that $\lceil \pi_t(j)/n^{1-s} \rceil = b$ and $\pi_t(j) > \ell$ we add 1 to $E[i, j]$. Notice that for each pair (i, t) this takes at most n^{1-s} time. Thus, E is created in $O(n^{3-s}) = O(n^{4-r})$ time.

The total running time of our algorithm is $O(n^{4-r} + n^{\omega(1,r,1)})$. Our choice of $r = 1.3165$ gives that $n^{4-r} = n^{2.6835}$. By Using $q = 7$ and $\beta = 0.033$ in Lemma 2.1 we get that $\omega(1, 1.3165, 1) < 2.6834$. Consequently, the overall running time is $O(n^{2.6835})$, as required. ∎

3 All Pairs Shortest Paths

Before proving Theorem 1.1, we need, once again, results concerning rectangular matrix multiplication. Before stating them we need to define two constants.

DEFINITION 3.1.

$$\alpha = \sup \{ 0 \le r \le 1 \mid \omega(1, r, 1) = 2 + o(1) \},$$
$$\beta = \frac{\omega - 2}{1 - \alpha}.$$

LEMMA 3.1. (COPPERSMITH [6]) $\alpha > 0.294$.

It is not difficult to see that Lemma 3.1 implies the following lemma. A proof can be found, for example, in [10].

LEMMA 3.2.

$$\omega(1, r, 1) \le \begin{cases} 2 + o(1) & \text{if } 0 \le r \le \alpha, \\ 2 + \beta(r - \alpha) + o(1) & \text{otherwise.} \end{cases}$$

A major part of the algorithm is based upon the following lemma. We first need a definition.

DEFINITION 3.2. Define μ to be the solution to $\omega(1 + \mu, 1, 1 + \mu) = 3$. By Lemma 3.2, $\mu > 0.338$.

One applies Lemma 3.2 by using the fact that $\omega(1 + \mu, 1, 1 + \mu) = \omega(1, r, 1)/r$ where $r = 1/(1 + \mu)$ and

solving $2 + \beta(r - \alpha) = 3r$, which yields $r = 0.747$. Notice that without Lemma 3.2 one can only trivially get $\mu \ge (3 - \omega)/2 \ge 0.312$, so, indeed, rectangular matrix multiplication turns out to be quite beneficial in this case.

Matrices over $R \cup \{\infty\}$ are called *distance matrices*. Recall that the *distance product* $C = A \star B$ of two $n \times n$ distance matrices A and B is defined by $C(i, j) = \min_{k=1}^{n} A(i, k) + B(k, j)$.

LEMMA 3.3. *Let A and B be two distance matrices, where each row of A contains at most n^ρ distinct values, and $\rho < \mu$. Then $A \star B$ can be computed in $O(n^{\omega(1+\rho, 1, 1+\gamma)})$ time where γ is the solution to the equation $\omega(1 + \rho, 1, 1 + \gamma) = 3 + \rho - \gamma$.*

Proof: We reduce the problem of computing $A \star B$ to the problem of computing a Boolean product of two large rectangular matrices. Set $k = n^\rho$ and $s = n^\gamma$.

For each row u of A, let $W(u) = \{w_{u1}, \ldots, w_{uk_u}\}$ be the set of distinct values appearing in row u. Let A' be the Boolean matrix of order $nk \times n$ indexed by the row set $[k] \times [n]$ and the column set $[n]$. We set $A'((j, u), v) = 1$ if $A(u, v) = w_{uj}$. Otherwise, $A'((j, u), v) = 0$. Notice that A' is constructed from A in $\Theta(n^2 k)$ time.

Let π_u be a permutation so that $\pi_u(i) < \pi_u(j)$ implies $B(i, u) \le B(j, u)$. We can construct $\pi_1, \ldots, \pi_n$ in $O(n^2 \log n)$ time by sorting each column of B. Let B' be the Boolean matrix of order $n \times ns$ indexed by the row set $[n]$ and the column set $[s] \times [n]$. We set $B'(u, (j, v)) = 1$ if $\pi_v(u) \in [1 + (j - 1)(n/s), \ldots, jn/s]$. Otherwise, $B'(u, (j, v)) = 0$. Notice that B' is constructed from the permutations in $\Theta(n^2 s)$ time.

Next, we compute $C' = A'B'$ in $O(n^{\omega(1+\rho, 1, 1+\gamma)})$ time. Let us now see how to deduce $D = A \star B$ from C'. Fixing u and v, we show how to compute $D(u, v)$ in $O(k(s + n/s))$ time. For each $i = 1, \ldots, k$, let j_i be the smallest index so that $C'((i, u), (j_i, v)) = 1$. It takes $O(s)$ time to locate j_i, and if no such index exists, we set $j_i = 0$. If $j_i \ne 0$, we know that we have at least one index z so that $A'((i, u), z) = 1$ and $B'(z, (j_i, v)) = 1$. In particular, this means that $\pi_v(z) \in [1 + (j_i - 1)(n/s), \ldots, j_i n/s]$. We wish to locate that z for which $\pi_v(z)$ is minimal. Since we only have to look at an interval of size n/s, we use π_v to locate z in $O(n/s)$ time. Denote the located z by z_i. In particular, we know that $A(u, z_i) + B(z_i, v)$ is the minimum sum whenever the first term has weight w_{ui}. Doing this for each $i = 1, \ldots, k$ and taking the minimum of $A(u, z_i) + B(z_i, v)$ ranging over all plausible i (for which $j_i \ne 0$), we obtain $D(u, v)$.

The total running time required to compute $D(u, v)$ is $O(k(s + n/s))$. Hence, D can be computed in

$O(n^2 k(s + n/s))$ time. Recall that

$$O(n^2 k(s + n/s)) = O(n^{2+\rho+\gamma} + n^{3+\rho-\gamma}) = O(n^{3+\rho-\gamma})$$

where the last equality follows from the fact that $\gamma \leq 0.5$. By our assumption on γ, we also have $O(n^{3+\rho-\gamma}) = O(n^{\omega(1+\rho,1,1+\gamma)})$ and hence the total running time of the algorithm is as claimed. $\blacksquare$

Proof of Theorem 1.1: For the rest of this section we assume that $G = (V, E, w)$ is a weighted directed graph with n vertices, and for each $v \in V$, $W(v)$ is the set of distinct weights of edges emanating from v. We assume that $|W(v)| \leq k$ for each $v \in V$. We construct an algorithm that computes the APSP distance matrix $D = D(G)$.

Let t be a positive integer parameter to be chosen later. For two vertices $u, v \in V$ let $c(u, v)$ denote the smallest possible number of edges on a shortest path from u to v. We define $c(u, u) = 0$ (we assume there are no negative weight cycles as these can easily be detected by the algorithm). Let $d_i(u, v)$ be the shortest distance from u to v that is obtained by paths consisting of at most i edges. Clearly, $d(u, v) = d_{n-1}(u, v)$. Also, for $u \neq v$, $d_1(u, v) = w(u, v)$ if $(u, v) \in E$ and otherwise $d_1(u, v) = \infty$. Finally, let D_i be the $n \times n$ matrix that records all the values $d_i(u, v)$. Thus, D_1 is easily constructed in $O(n^2)$ time. The diagonal of D_1, as well as all other D_i, contains only zeros.

Notice that D_i is obtained as the distance product $D_1 \star D_{i-1}$. This is obvious, as each shortest path from u to v consisting of at most i edges, is a combination of its first edge (u, x), and its remaining path which is a shortest path from x to v containing at most $i-1$ edges. It is also important to note that each row of D_1 contains at most k distinct values. Thus, putting $k = n^\rho$, we obtain from Lemma 3.3 that D_i can be constructed in $O(n^{\omega(1+\rho,1,1+\gamma)})$ time where γ is the solution to the equation $\omega(1 + \rho, 1, 1 + \gamma) = 3 + \rho - \gamma$.

We sequentially compute D_i for all $i = 1, \ldots, t$ in $O(tn^{\omega(1+\rho,1,1+\gamma)}) = O(tn^{3+\rho-\gamma})$ time. After obtaining D_t, we switch to the second part of the algorithm, which no longer involves fast matrix multiplication.

We already know that $D_t(u, v) = D(u, v)$ for all pairs with $c(u, v) \leq t$. We thus need to worry about those pairs for which $c(u, v) > t$. For this, we shall use the bridging sets techniques of Zwick [16].

DEFINITION 3.3. *A set $B \subset V$ is a t-bridging set if for each $u, v \in V$ with $\infty > c(u, v) > t$, there exists a shortest path $p = \{u = u_0, u_1, \ldots, u_s = v\}$ realizing $d_{\lfloor 3t/2 \rfloor}(u, v)$ where some vertex u_i on p is from B and $i \leq t$ and $s - i \leq t$.*

It is quite easy to show (see [16]) that a *random* set B of vertices of size $\Theta(n \log n/t)$ is a t-bridging

set. In fact, Zwick has shown how to find such a B *deterministically* using D_t, in $O(n^{2.5})$ time (more precisely, a t-bridging set of size $\max\{\sqrt{n}, n \log n/t\}$ can be found deterministically in $O(n^{2.5})$ time).

Having found a t-bridging set B, we compute a matrix which, in a sense, is better than $D_{(3/2)t}$, as follows (although we should write $\lfloor 3t/2 \rfloor$ we ignore floors an ceilings for clarity). Take the sub-matrix of $D(B)_t$ of D_t consisting of all the rows but only the columns belonging to B. Now compute $C = D(B)_t \star D(B)_t^T$ in the straightforward way in $\tilde{O}(n^3/t)$ time. By the definition of B, in the resulting distance product $C(u, v) \leq D_{(3/2)t}$. In particular, we have $C(u, v) = D(u, v)$ for all pairs with $c(u, v) \leq 3t/2$. Repeating the same squaring process a logarithmic number of times, we eventually obtain D. Notice that in iteration i we obtain a distance matrix which is better than $D_{(3/2)^i t}$. In fact, notice that we do not need to take smaller and smaller bridging sets as the process continues; we can do with bridging sets of size $\Theta(n \log n/t)$ throughout all the iterations. The overall running time would still be $\tilde{O}(n^3/t)$.

Summing up the running times of the two parts of the algorithm, the overall running time is $\tilde{O}(tn^{3+\rho-\gamma} + n^3/t)$. Choosing $t = n^{(\gamma-\rho)/2}$ we obtain that the overall running time of the algorithm is $\tilde{O}(n^{3-\gamma/2+\rho/2})$, as required. $\blacksquare$

We have not shown how to compute the actual shortest paths yielding the shortest distances. However, this is a standard procedure which is based upon *witnesses* for Boolean matrix multiplication [2, 9] and that has been used in all shortest paths algorithms that involve fast matrix multiplication (see, e.g., [16]). Full details will appear in the full version.

4 Expanding sets of permutations

In this section we prove Theorem 1.2. For the rest of this section we assume $P \subset S_n$ has size $|P| = p$ and is given in a $p \times n$ array.

With each $\pi \in S_n$ we associate a *signature* $s(\pi)$ which is the bilinear expression defined by

$$s(\pi) = \sum_{i=1}^{n} x_i y_{\pi(i)}.$$

Hence, for example, for $(2341) \in S_4$ we have $s(2341) = x_1 y_2 + x_2 y_3 + x_3 y_4 + x_4 y_1$. It is obvious, but important, to note that indeed $s(\pi)$ is a signature. Namely, if $\pi \neq \pi'$, then $s(\pi) \neq s(\pi')$.

For a permutation π the *characteristic x-vector* of π is the vector $x(\pi) = (x_{\pi(1)}, \ldots, x_{\pi(n)})$. Similarly, the characteristic y-vector is $y(\pi) = (y_{\pi(1)}, \ldots, y_{\pi(n)})$.

The following identity, whose proof is straightforward, shows how to compute the signature of a product of two permutations using the inner product of two characteristic vectors.

$$(4.1) \qquad s(\pi_1\pi_2) = x(\pi_1^{-1})y(\pi_2)^T.$$

Indeed, consider a term x_iy_j in $s(\pi_1\pi_2)$. This term appears since $\pi_2(\pi_1(i)) = j$. Let $\pi_1(i) = k$. Then $\pi_1^{-1}(k) = i$. Thus, the k'th coordinate of $x(\pi_1^{-1})$ is x_i. Similarly, the k'th coordinate of $y(\pi_2)$ is y_j. Thus, x_iy_j is also a term in the inner product $x(\pi_1^{-1})y(\pi_2)^T$. The converse is equally true as well.

As a concrete example, consider two permutations of S_6, $\pi_1 = (641352)$ and $\pi_2 = (425631)$. Then, $\pi_1\pi_2 = (164532)$, $\pi_1^{-1} = (364251)$,

$$
\begin{aligned}
x(\pi_1^{-1}) &= (x_3, x_6, x_4, x_2, x_5, x_1) \\
y(\pi_2) &= (y_4, y_2, y_5, y_6, y_3, y_1)
\end{aligned}
$$

and indeed,

$$
\begin{aligned}
&x(\pi_1^{-1})y(\pi_2)^T \\
=\ & x_3y_4 + x_6y_2 + x_4y_5 + x_2y_6 + x_5y_3 + x_1y_1 \\
=\ & s(\pi_1\pi_2).
\end{aligned}
$$

As defined in the introduction, for a positive integer j, let P^j be the set of all permutations obtained as a product of a sequence of j elements of P. We also define $P^{-1} = \{\pi^{-1} : \pi \in P\}$ and $P^{-j} = (P^{-1})^j$. Notice that $(P^j)^{-1} = (P^{-1})^j$, $|P| = |P^{-1}| = p$, and $|P^j| = |P^{-j}| \leq p^j$. Finally, for $Q \subset S_n$ we define $S(Q) = \{s(\pi) : \pi \in Q\}$.

LEMMA 4.1. *For any $1 \leq j \leq k - 1$, P is k-free if and only if $S(P^j) \cap S(P^{j-k}) = \emptyset$.*

Proof: As signatures identify permutations uniquely, it suffices to prove that P is k-free if and only if $P^j \cap (P^{k-j})^{-1} = \emptyset$. Indeed, if P is not k-free, then $\pi_1 \cdots \pi_k = id$ for some sequence of k permutations. Hence, $\pi_1 \cdots \pi_j = \pi_k^{-1} \cdots \pi_{j+1}^{-1}$. But $\pi_1 \cdots \pi_j \in P^j$ and $\pi_k^{-1} \cdots \pi_{j+1}^{-1} \in (P^{k-j})^{-1}$. The converse is equally true as well. ∎

COROLLARY 4.1. *P is k-free if and only if $S(P^{\lceil k/2 \rceil}) \cap S(P^{-\lfloor k/2 \rfloor}) = \emptyset$.*

The correctness of our algorithm is based upon Corollary 4.1. We will efficiently hash all the element in $S(P^{\lceil k/2 \rceil})$ in a set A and efficiently hash all the elements in $S(P^{-\lfloor k/2 \rfloor})$ in a set B. We will then quickly check if $A \cap B = \emptyset$. If so, we conclude that P is k-free. Otherwise, using the properties of our hashing, this will

imply that P is not k-free whp, and we will be able to produce a witness to this fact.

The algorithm: We set $t_1 = \lceil (k-2)/4 \rceil$, $t_2 = \lceil k/4 \rceil$, and notice that $t_1 + t_2 = \lceil k/2 \rceil$. We compute P^{t_1} in $O(p^{t_1}n)$ time in the obvious way; for each possible sequence of t_1 elements of P, their product is computed in $O(t_1 n) = O(n)$ time. Notice that we can avoid multiplicities in P^{t_1} using radix sort (this is not crucial, however). We compute P^{-1} in $O(pn)$ time and using P^{-1} we compute P^{-t_2} in $O(p^{t_2}n)$ time. We note that for $k \in \{3, 4\}$ we actually have $t_1 = t_2 = 1$ so in these cases there is nothing to compute but P^{-1}, as $P^{t_1} = P$ and $P^{-t_2} = P^{-1}$ in these cases.

For $Q \subset S_n$, let $X(Q)$ be the $|Q| \times n$ matrix where $X(\pi, i) = x_{\pi(i)}$. Namely, the row corresponding to π is $x(\pi)$, the characteristic x-vector of π. Similarly, let $Y(Q)$ be the $|Q| \times n$ matrix where $Y(\pi, i) = y_{\pi(i)}$. Namely, the row corresponding to π is $y(\pi)$. Using the previously computed P^{t_1} and P^{-t_2} we can now show the following.

LEMMA 4.2. *The set of entries of the matrix product $X(P^{-t_2})Y(P^{t_1})^T$ is precisely $S(P^{\lceil k/2 \rceil})$.*

Proof: Consider an element $\pi \in P^{\lceil k/2 \rceil}$ and its signature $s(\pi) \in S(P^{\lceil k/2 \rceil})$. Since $t_1 + t_2 = \lceil k/2 \rceil$ we can write π as $\pi = \pi_2\pi_1$ where $\pi_2 \in P^{t_2}$ and $\pi_1 \in P^{t_1}$. As $\pi_2^{-1} \in P^{-t_2}$ there is a row of $X(P^{-t_2})$ corresponding to $x(\pi_2^{-1})$. Similarly, there is a row of $Y(P^{t_1})$ (which is a column of $Y(P^{t_1})^T$) corresponding to $y(\pi_1)$. Thus, in the matrix product, there is an element corresponding to the inner product $x(\pi_2^{-1})y(\pi_1)^T$. By (4.1), this inner product is just $s(\pi_2\pi_1) = s(\pi)$. The converse is equally true as well. ∎

We set $t_3 = \lfloor k/4 \rfloor$, $t_4 = \lfloor (k+2)/4 \rfloor$, and notice that $t_3 + t_4 = \lfloor k/2 \rfloor$. We compute P^{t_4} in $O(p^{t_4}n)$ time and P^{-t_3} in $O(p^{t_3}n)$ time as in the case of t_1 and t_2 shown earlier. Notice that if $k = 3$, then $t_3 = 0$, so $P^{-t_3} = P^0 = \{id\}$ in this case. As in Lemma 4.2 we analogously have that:

LEMMA 4.3. *The set of entries of the matrix product $X(P^{t_4})Y(P^{-t_3})^T$ is precisely $S(P^{-\lfloor k/2 \rfloor})$.*

Using Corollary 4.1 we have that by computing both products $Z = X(P^{-t_2})Y(P^{t_1})^T$ and $W = X(P^{t_4})Y(P^{-t_3})^T$ we can determine, by examining the entries of Z and W, whether P is k-free. Clearly, if we find two equal entries in Z and W we can immediately obtain a witness to the non k-freeness of P. Unfortunately, computing Z and W is time consuming as these are matrices over the ring of polynomials (each is a bilinear term) with $2n$ indeterminates. Fortunately, however, by paying with randomization, we can reduce the

computation time considerably. To show this, we need the following result of Zippel [15] and Schwartz [12].

LEMMA 4.4. *If $f(x_1, \ldots, x_n, y_1, \ldots, y_n)$ is a non-zero polynomial of degree d with integer coefficients and $R = \{1, \ldots, r\}$, then the probability that f equals zero on a random element $(c_1, \ldots, c_{2n}) \in R^{2n}$ is at most d/r.*

We choose $r = \Theta(p^{k+1})$ and assign to each variable x_i and each variable y_j an integer from $\{1, \ldots, r\}$ selected uniformly and independently at random. We replace each occurrence of each indeterminate in $X(P^{t_4})$, $Y(P^{-t_3})^T$, $X(P^{-t_2})$, and $Y(P^{t_1})^T$ with its corresponding assigned random value. Denote the resulting random matrices by $X_R(P^{t_4})$, $Y_R(P^{-t_3})^T$, $X_R(P^{-t_2})$, and $Y_R(P^{t_1})^T$, respectively. We can now show:

LEMMA 4.5. *With high probability, there is a common entry in $Z_R = X_R(P^{-t_2})Y_R(P^{t_1})^T$ and in $W_R = X_R(P^{t_4})Y_R(P^{-t_3})^T$ if and only if there is a common entry in $Z = X(P^{-t_2})Y(P^{t_1})^T$ and in $W = X(P^{t_4})Y(P^{-t_3})^T$.*

Proof: One direction is trivial. If there is an entry common to Z and W, then there is also a common entry in Z_R and W_R since the latter are obtained by consistently replacing each indeterminate with specific values. Let T be the set of all distinct entries in Z and in W. Notice that T consists of at most $2p^{\lceil k/2 \rceil}$ nonzero polynomials (each is a signature of some permutation). The probability that a pair of elements of T are equal after replacing the x's and the y's with the random integers is at most $2/r$, by Lemma 4.4 (for each distinct pair f_1, f_2 of elements of T we have that the polynomial $f_1 - f_2$ is non-zero and has degree 2). As there are less than $|T|^2 \le 4p^{k+1}$ possible pairs, and as $r = \Theta(p^{k+1})$, the probability that the elements of T are replaced with distinct integers is very high (say, higher than 0.99). The result follows. ∎

By Lemma 4.5 our algorithm is completed by checking for common entries in Z_R and W_R. The most time consuming part of the algorithm is the computation of the matrix products yielding Z_R and W_R. Using fast matrix multiplication, and setting $p = n^\beta$ we have that Z_R can be computed using $O(n^\alpha)$ arithmetic operations where $\alpha = \omega(\beta\lceil (k-2)/4 \rceil, 1, \beta\lceil k/4 \rceil)$. Each element in $X_R(P^{-t_2})$ and in $Y_R(P^{t_1})^T$ is an integer not exceeding $nr^2 = \Theta(n^{2k\beta+1})$ and hence has only $\Theta(\log n)$ bits. Thus the actual running time required to compute Z_R is also $O(n^\alpha)$. The same holds for the time required to compute W_R (notice that $t_3 \le t_1$ and $t_4 \le t_2$). Consequently, the overall running time of the algorithm is $O(n^\alpha)$, completing the proof of Theorem 1.2. ∎

5 Concluding remarks and open problems

An interesting open problem is whether our bound of $n^{0.338}$ possible distinct weights emanating from each vertex, which suffices for a truly sub-cubic APSP algorithm, can be improved upon (without improving matrix multiplication exponents). Clearly, if one can show that $\Theta(n)$ distinct weight are allowed this would solve the longstanding open problem of computing a distance product in truly sub-cubic time. Still, any improvement over the $\Theta(n^{0.338})$ bound would be of interest, and importance.

We have also shown an efficient algorithm for detecting fully k-expansion or k-freeness of a set of permutations. However, our algorithm is randomized. Perhaps the first interesting open problem is to find a sub-cubic *deterministic* algorithm for deciding, say, if a set of n permutations of S_n is fully 2-expanding.

References

[1] N. Alon, *Generating pseudo-random permutations and maximum-flow algorithms*, Information Processing Letters, 35 (1990), pp. 201–204.

[2] N. Alon and M. Naor, *Derandomization, witnesses for Boolean matrix multiplication and construction of perfect hash functions*, Algorithmica, 16 (1996), pp. 434–449.

[3] A. Z. Broder, M. Charikar, and M. Mitzenmacher, *A derandomization using min-wise independent permutations*, Journal of Discrete Algorithms, 1 (2003), pp. 11–20.

[4] T. M. Chan, *All Pairs Shortest Paths with Real Weights in $O(n^3/\log n)$ Time*, Proceedings of the 9th Workshop on Algorithms and Data Structures (WADS), Lecture Notes in Computer Science, 3608 (2005), pp. 318–324.

[5] T. M. Chan, *More Algorithms for All-Pairs Shortest Paths in Weighted Graphs*, Proceedings of the 39th ACM Symposium on Theory of Computing (STOC), ACM Press (2007), pp. 590–598.

[6] D. Coppersmith, *Rectangular matrix multiplication revisited*, Journal of Complexity, 13 (1997), pp. 42–49.

[7] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progressions*, Journal of Symbolic Computation, 9 (1990), pp. 251–280.

[8] M. L. Fredman, *New bounds on the complexity of the shortest path problem*, SIAM Journal on Computing, 5 (1976), pp. 49–60.

[9] Z. Galil and O. Margalit, *Witnesses for Boolean matrix multiplication and for transitive closure*, Journal of Complexity, 9 (1993), pp. 201–221.

[10] X. Huang and V. Y. Pan, *Fast rectangular matrix multiplications and applications*, Journal of Complexity, 14 (1998), pp. 257–299.

[11] J. Matousek, *Computing dominances in E^n*, Information Processing Letters, 38 (1991), pp. 277–278.

[12] J. T. Schwartz, *Fast Probabilistic Algorithms for Verification of Polynomial Identities*, Journal of the ACM, 27 (1980), pp. 701–717.

[13] V. Vassilevska and R. Williams, *Finding a maximum weight triangle in $n^{3-\delta}$ time, with applications*, Proceedings of the 38th ACM Symposium on Theory of Computing (STOC), ACM Press (2006), pp. 225–231.

[14] V. Vassilevska, R. Williams, and R. Yuster, *All pairs bottleneck paths for general graphs in truly sub-cubic time*, Proceedings of the 39th ACM Symposium on Theory of Computing (STOC), ACM Press (2007), pp. 585–589.

[15] R. Zippel, *Probabilistic algorithms for sparse polynomials*, Proceedings of Symbolic and Algebraic Computation (EUROSAM), Lecture Notes in Computer Science 72 (1979), pp. 216–226.

[16] U. Zwick, *All-pairs shortest paths using bridging sets and rectangular matrix multiplication*, Journal of the ACM, 49 (2002), pp. 289–317.