Division in the Plactic Monoid

Chris Monico

Department of Mathematics and Statistics
Texas Tech University
e-mail: c.monico@ttu.edu

Draft of December 2, 2022

Abstract

In [1], a novel cryptographic key exchange technique was proposed using the plactic monoid, based on the apparent difficulty of solving division problems in that monoid. Specifically, given elements c, b in the plactic monoid, the problem is to find q for which $qb = c$, given that such a q exists. In this paper, we introduce a metric on the plactic monoid and use it to give a probabilistic algorithm for solving that problem which is fast for parameter values in the range of interest.

1 Introduction

For two parties, Alice and Bob, wishing to communicate securely over an insecure channel, the principal difficulty is agreeing on a shared secret key to be used in a cipher. In the mid-to-late 1970's, several potential solutions to this problem were given which have given rise to the primary solutions that are still in use today. However, in 1994 Peter Shor [6] gave a *quantum algorithm* capable of rendering those cryptosystems insecure on a sufficiently robust quantum computer. Since that time, some progress has been made toward building such a quantum computer, and so it has become increasingly important to consider new potential solutions, with the hope of finding one which would remain secure even against an adversary with such a device.

The present paper is concerned with studying one such recent proposal. In [1], Daniel Brown proposed a very interesting and novel key agreement method which may be considered as an instance of Rabi and Sherman's key agreement protocol [4], but using the associative operation provided by the plactic monoid, $\mathcal{P}_N$. There is substantial literature on the plactic monoid, but we will give a brief description in the next section. For the time being, it is enough to know that $\mathcal{P}_N$ is a *monoid*, or semigroup with identity. That is, $\mathcal{P}_N$ is a set with an associative operation and an identity element. The proposed method in [1] is as follows:

1. Alice and Bob agree on an element $g \in \mathcal{P}_N$.
2. Alice privately chooses $a \in \mathcal{P}_N$, computes $\alpha = ag$, and sends α to Bob.

3. Bob privately chooses $b \in \mathcal{P}_N$, computes $\beta = gb$, and sends β to Alice.

4. Since the operation in $\mathcal{P}_N$ is associative, both Alice and Bob can compute the shared secret key $\kappa = agb = a\beta = \alpha b$.

In this case, an eavesdropper Eve would know $\mathcal{P}_N, g, \alpha$, and β. If she could determine an element $a' \in \mathcal{P}_N$ for which $a'g = \alpha$, she could compute $a'\beta = a'(gb) = (a'g)b = \alpha b = \kappa$. Or, she could similarly recover κ if she could find an element $b' \in \mathcal{P}_N$ for which $gb' = \beta$. So the security of this system rests on the apparent difficulty of this problem.

The proposal above is very clever, and the eavesdropper's problem does at first appear to be very formidable. Brown himself considered many possible approaches in [1], all of which seem to require a number of operations which is exponential in the sizes of a, g, and b. We also considered several other approaches very different from those in [1], all of which apparently required an exponential number of operations, before discovering the technique described in this paper.

The algorithm we will propose here is probabilistic and we have been thus far unable to rigorously analyze the runtime, though it is plausibly polynomial-time for each fixed N. Even if it is not polynomial-time, the algorithm would still seem to be of interest since it was able to solve the challenge problem in [1]. Our estimate is that the previously best known method would have required around 10^{20} CPU-years with current technology, whereas the present method recovered a solution in around 10 CPU-minutes.

In Section 2, we will give the necessary background on the plactic monoid, before introducing a metric on $\mathcal{P}_N$ in Section 3. Section 4 describes a first approach to solve the eavesdropper's problem, which forms the basis for the apparently faster method given in Section 5.

2 Background on the Plactic Monoid

Let N be a positive integer, and $\mathcal{A}_N$ be a set of N symbols together with an ordering. For convenience, we will assume that $\mathcal{A}_N = \{1, 2, \ldots, N\}$, but the elements are formal symbols, not to be interpreted as integers. That is, we are considering $\mathcal{A}_N$ as an ordered alphabet. The free monoid $\mathcal{A}_N^*$ generated by $\mathcal{A}_N$ is the collection of all finite-length words in the alphabet $\mathcal{A}_N$, including the empty word, with the operation of concatenation. For a word $w \in \mathcal{A}_N^*$ we let $|w|$ denote its length, and $w_j \in \mathcal{A}_N$ the j-th symbol in the word w, so that $w = w_1 w_2 \ldots w_{|w|}$.

A *semistandard Young tableau* over $\mathcal{A}_N$ is a finite sequence $T = (R_1, R_2, \ldots, R_N)$ of (possibly empty) words in $\mathcal{A}_N^*$ subject to the following constraints:

(i) $|R_1| \geq |R_2| \geq \cdots \geq |R_N|$,

(ii) the symbols of each nonempty word R_i appear in non-decreasing order,

(iii) if $1 \leq i < N$ and $1 \leq j \leq |R_{i+1}|$, then $(R_{i+1})_j > (R_i)_j$; that is, the j-th symbol of R_{i+1} is strictly greater than the j-th symbol of R_i.

This is most easily understood visually, with the sequence $R_1, \ldots, R_N$ of words represented in a left-aligned grid, as in Figure 1. In this paper, we use the so-called 'French notation', with R_1 written at the bottom, R_2 above R_1, and so on.

4	4	5		
2	3	4		
1	1	1	2	3

Figure 1: A semistandard Young tableau over $\mathcal{A}_5$ with rows $R_1 = 11123$, $R_2 = 234$, $R_3 = 445$, and R_4, R_5 empty.

In terms of this visual representation, the three conditions above assert that (i) the rows are of non-decreasing length, from top to bottom, (ii) the entries in each row form a non-decreasing sequence, from left to right, and (iii) the entries in each column form a strictly decreasing sequence, from top to bottom.

In this paper, all tableaux under consideration are semistandard Young tableaux, so we will refer to them simply as *tableaux*. The *row reading* of a tableau $T = (R_1, \ldots, R_N)$, denoted $\rho(T)$, is the word in $\mathcal{A}_N^*$ obtained by concatenation of $R_N, R_{N-1}, \ldots, R_1$,

$$\rho(T) = R_N R_{N-1} \ldots R_1.$$

For example, if T is the tableau in Figure 1 then $\rho(T) = 44523411123$.

For $w \in \mathcal{A}_N^*$, Schensted's Insertion Algorithm [5] constructs a tableau corresponding to w. If w is the empty word, then $P(w)$ is the empty tableau, and the construction then proceeds inductively. Given the tableau $P(w)$ and a symbol $s \in \mathcal{A}_N$, the tableau $P(ws)$ is found as follows:

1. Set $r \leftarrow 1$, $T \leftarrow P(w)$, and $x \leftarrow s$.

2. If appending x to row r of T would yield a sorted row, then do so and goto Step 4.

3. Find the leftmost entry of row r of T which is greater or equal x; set y to be the value of that entry, replace the entry with x, then set $x \leftarrow y$, $r \leftarrow r + 1$ and goto Step 2 (x 'bumps' the entry y up to the next row).

4. T is the tableau $P(ws)$ of the word ws.

For example, consider $w = 44523411123$ which has the tableau $P(w)$ given in Figure 1, and let $s = 2$. To determine the tableau $P(ws)$, we follow the procedure described above. First set $r \leftarrow 1$ and $x \leftarrow s = 2$. Iterating as described above, we obtain the sequence of values for T, r and x shown in Figure 2 just before each execution of Step 2.

The map P from $\mathcal{A}_N^*$ to the set of all semistandard young tableau is not injective. But Knuth [2] precisely characterized the words having the same image under P. Define the *Knuth relations* as follows: for all $a, b, z \in \mathcal{A}_N$ we define

$$\begin{aligned} zab &\sim bza, \quad \text{if } \min\{a,b\} < z \leq \max\{a,b\}, \\ abz &\sim baz, \quad \text{if } \min\{a,b\} \leq z < \max\{a,b\}. \end{aligned}$$

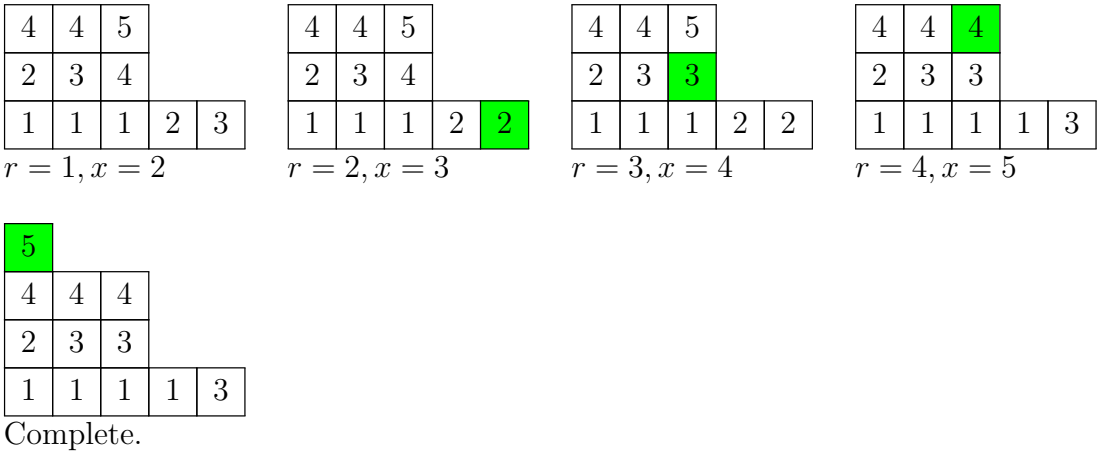

Figure 2: Computing the tableau $P(445234111232)$ from $P(44523411123)$. Shown is the sequence of values of T, r and x immediately before each execution of Step 2.

These relations generate a congruence relation $\sim$ on $\mathcal{A}_N^*$, and the *plactic monoid* on $\mathcal{A}_N$ is defined to be the quotient $\mathcal{P}_N = \mathcal{A}_N^*/\sim$. For a word $w \in \mathcal{A}_N^*$, we let $[w]$ denote the equivalence class of w in $\mathcal{P}_N$. Knuth showed [2, Theorem 6] that $P(u) = P(v)$ if and only if $[u] = [v]$. Moreover, it's not hard to see that $P(\rho(T)) = T$ for every tableaux T. This provides a mechanism for efficiently determining a canonical representative for each equivalence class $[u] \in \mathcal{P}_N$: one may take $\rho(P(u))$ as the canonical representative of $[u]$.

3 A metric on $\mathcal{P}_N$

For $w \in \mathcal{A}_N^*$, and a positive integer r, let $P_r(w) \in \mathcal{A}_N^*$ be the r-th row of the tableau $P(w)$; and for $s \in \mathcal{A}_N$, let $|w|_s$ be the number of copies of the symbol s appearing in w. Then for $u, v \in \mathcal{A}_N^*$ we define

$$d(u,v) = \frac{1}{2} \sum_r \sum_s \Big| |P_r(u)|_s - |P_r(v)|_s \Big|.$$

Proposition 3.1. *Let N be a positive integer. The function d defined above is a pseudometric on $\mathcal{A}_N^*$. Moreover, for words $u, v \in \mathcal{A}_N^*$ of the same length, $d(u,v)$ is integer-valued.*

Proof. It's clear that d is non-negative, and $d(u,u) = 0$.

Since $||P_r(u)|_s - |P_r(v)|_s| = ||P_r(v)|_s - |P_r(u)|_s|$ for all r, s, we have that $d(u,v) = d(v,u)$. A standard application of the triangle inequality for the absolute value shows that d satisfies the triangle inequality as well.

For the final claim, let $u, v \in \mathcal{A}_N^*$ with $|u| = |v|$. Certainly $2d(u,v) \in \mathbb{Z}$, so it suffices to

show that $2d(u,v)$ is an even integer. We have that

$$\begin{aligned} 2d(u,v) &= \sum_r \sum_s ||P_r(u)|_s - |P_r(v)|_s| \\ &\equiv \sum_r \sum_s (|P_r(u)|_s - |P_r(v)|_s) \pmod 2 \\ &\equiv \sum_r \sum_s |P_r(u)|_s - \sum_r \sum_s |P_r(v)|_s \\ &\equiv |u| - |v| \equiv 0 \pmod 2 . \end{aligned}$$

□

If $[u] = [v]$, then $P(u) = P(v)$, and so $P_r(u) = P_r(v)$ for every positive integer r. It follows that $d(u,x) = d(v,x)$ for all $x \in \mathcal{A}_N^*$, and so d is well-defined on $\mathcal{P}_N$ via

$$d([u],[w]) = d(u,w),$$

and it is easy to see that d is a metric on $\mathcal{P}_N$. It is a fairly natural metric, as it essentially counts the number of symbols in which the tableaux of $[u]$ and $[w]$ differ, so it seems possible that it has been used previously, but we were unable to find any reference.

For $w \in \mathcal{A}_N^*$ with $n = |w|$ and a permutation $\sigma \in S_n$, we let $\sigma(w)$ be the word obtained by permuting the symbols of w accordingly,

$$\sigma(w) = w_{\sigma(1)} w_{\sigma(2)} \dots w_{\sigma(n)}.$$

Proposition 3.2. *Let $[w] \in \mathcal{P}_N$. The number of $[x] \in \mathcal{P}_N$ for which $x = \sigma(w)$ for some $\sigma \in S_N$ and $d(w,x) = 1$ is at most $\sum_{s=1}^{N} (s-1) \min\{s, |w|_s\}$.*

Proof. Consider the tableau $P(w)$. Notice that $x = \sigma(w)$ for some $\sigma \in S_N$ and $d(w,x) = 1$ if and only if the tableau $P(x)$ is obtained by moving a single symbol from $P(w)$ to a different row. The symbol $s \in \mathcal{A}_N$ might appear in rows $1, 2, \dots, s$ of $P(w)$, but may not appear in rows $s+1, \dots, N$ and $|w|_s$ is the number of copies of s in $P(w)$. So there are $\min\{s, |w|_s\}$ ways in which one could choose a copy of s to move, and it could be moved to at most $s-1$ possible destinations to yield a tableau of distance 1 away from $P(w)$. □

The bound given by the proposition above will play a role in the algorithm described in the next section. A small improvement is useful, though. If w is a word which does not contain all of the symbols in $\mathcal{A}_N$, it could be embedded into $\mathcal{A}_M$ for some $M < N$, in which case the result could be strengthened. For $w \in \mathcal{A}_N^*$, let $\operatorname{supp}(w)$ be the set of distinct symbols appearing in w and $f_s(w) = |\operatorname{supp}(w) \cap \{1, 2, \dots, s\}|$. Setting

$$D(w) = \sum_{s=1}^{N} (f_s(w) - 1) \min\{f_s(w), |w|_s\}, \tag{3.1}$$

we obtain a more general bound on the number of such $[x]$ with $d(w,x) = 1$.

4 First approach

In this section, we will sketch a first approach to solving the following computational problem. It forms the basis for the apparently faster modified version with lifting in the next section.

Problem 4.1 (Division Problem in $\mathcal{P}_N$)**.** *For $[b], [c] \in \mathcal{P}_N$, find $[q] \in \mathcal{P}_N$ such that $[q][b] = [c]$, given that such a $[q]$ exists.*

Suppose that $[b], [c] \in \mathcal{P}_N$ and that $[q][b] = [c]$ for some q. Let $q_0 \in \mathcal{A}_N^*$ be an arbitrary word for which $|q_0|_s = |c|_s - |b|_s$ for each $s \in \mathcal{A}_N$. By assumption, there is at least one permutation $\sigma \in S_{|q_0|}$ for which $[\sigma(q_0)][b] = [\sigma(q_0)b] = [c]$. In general, since $\mathcal{P}_N$ is non-cancellative, there may be several such σ. Simply enumerating them and testing one at a time is out of the question, of course, since there are $|q_0|!$ such permutations. Instead, we adopt what Daniel Brown has observed is essentially a hill-climbing approach, by trying to find a sequence $\sigma_1, \sigma_2, \ldots, \sigma_k$ of permutations for which

$$q_j = \sigma_j(q_{j-1}), \qquad \text{and} \quad d(q_j b, c) < d(q_{j-1} b, c).$$

Since $d(q_j b, c)$ is integer-valued, we would have $d(q_j b, c) \leq d(q_{j-1} b, c) - 1 \leq d(q_0 b, c) - j$, so that we would obtain a solution $d(q_t b, c) = 0$ for some $t \leq d(q_0 b, c) \leq |c|$. The central idea here is trying to efficiently find such a sequence of permutations.

A first attempt at solving the division problem might proceed as follows. Let $\Sigma_n \subset S_n$ be a relatively small collection of permutations σ for which $d(\sigma(q), q)$ is typically small and positive. For some such permutations, $d(\sigma(q)b, qb)$ might be large, but hopefully it will often be small and positive. Then proceed as follows.

1. Set q_0 to be a randomly chosen word with $|q_0|_s = |c|_s - |b|_s$ for all $s \in \mathcal{A}_N$. Set $k \leftarrow 0$ and $n \leftarrow |q_0|$.

2. While $d(q_k b, c) > 0$ do the following:

   (i) Randomly choose a permutation $\sigma \in \Sigma_n$.

   (ii) If $d(\sigma(q_k)b, c) < d(q_k b, c)$, then set $q_{k+1} \leftarrow \sigma(q_k)$ and $k \leftarrow k + 1$.

The collection Σ_n of permutations we will use is as follows. For distinct integers $i, j \in \{1, 2, \ldots n\}$ with $i < j$, let σ_{ij} be the cycle $(j, j-1, j-2, \ldots, i+1, i) \in S_n$, and let $\sigma_{ji} = \sigma_{ij}^{-1}$. Let $\Sigma_n = \{\sigma_{ij} \ : \ i, j \in \{1, 2, \ldots, n\}\}$. A permutation σ_{ij} of this form applied to a word w of length n essentially deletes the symbol at position i and re-inserts it at position j; in other words, it 'slides' the symbol at position i to position j. It's easy to see that every permutation in S_n can be obtained as a composition of at most n such permutations in Σ_n.

This idea forms the basis for our approach, but the technique described above will often not converge to a solution. The principal obstruction is that it can happen that $d(qb, c) > 0$ and $d(\sigma(q)b, c) \geq d(qb, c)$ for all $\sigma \in \Sigma_n$. In order to overcome this obstruction, we make two small modifications to the idea above.

The first modification is that at Step 2(ii), we relax the inequality and set $q_{k+1} \leftarrow \sigma(q_k)$ if $d(\sigma(q_k)b, c) \leq d(q_k b, c)$. For some values of q with $d(qb, c) = \delta$, the probability of finding

a σ which improves the distance may be much smaller than the same probability for other values of q' with $d(q'b, c) = \delta$. The idea is to let the sequence 'wander' around on the sphere of radius δ until it finds a permutation which strictly improves the distance.

The second modification is as follows. For some values of q it appears that the 'wandering' above may be limited to a subset of the sphere of radius δ for which the probabilities of improving distance are all unusually small (or perhaps even zero). If too much time is spent on a sphere of a given radius δ, we will take a 'jump' which might worsen the distance. The criterion we employ is that if we have likely visited most of the equivalence classes $[\sigma(q_k)b]$ with $\sigma \in \Sigma_n$ for which $d(\sigma(q_k)b, q_k b) = 1$, then we will take such a 'jump'. The number of such equivalence classes is not more than $D(q_k b)$; in fact, it is probably considerably less, since we use a restricted set of permutations and only permute half of the word $q_k b$.

The jump will begin from a 'best' value of q discovered so far; that is, a value of q which has so far resulted in the least value of $d(qb, c)$. Then apply a variable number (but at least two) randomly chosen permutations to q. The idea is to jump by at least two permutations, to lessen the likelihood of quickly returning to the same point. But it can happen that the sequence gets stuck at a local minimum, so as more jumps are taken the likelihood of farther jumps being considered increases. This is an ad-hoc solution, but does seem to work in practice. With these modifications, the approach is described below as Algorithm 1.

Algorithm 1 Plactic Division

Input: $c, b, q_0 \in \mathcal{A}_N^*$ such that $[c] = [\sigma(q_0)][b]$ for some permutation $\sigma \in S_{|q_0|}$.
Output: A value of $q \in \mathcal{A}_N^*$ satisfying $[c] = [q][b]$.

1. Set $q \leftarrow q_0$, $n \leftarrow |q|$, $m \leftarrow 0$, and $M = 2D(qb)$, the function D given by (3.1). Set $\delta_{\text{global}} \leftarrow d(qb, c)$ and $q_{\text{global}} \leftarrow q$.

2. While $d(qb, c) > 0$ do the following:

   (i) Randomly choose a permutation $\sigma \in \Sigma_n$, and set $\delta \leftarrow d(\sigma(q)b, c)$.

   (ii) If $d(\sigma(q)b, qb) = 1$ then set $m \leftarrow m + 1$. (count potential steps of distance 1)

   (ii) If $\delta < d(qb, c)$ then set $m \leftarrow 0$. (distance improved)

   (iii) If $\delta \leq d(qb, c)$, then set $q \leftarrow \sigma(q)$. (take this step)

   (iv) If $\delta \leq \delta_{\text{global}}$, set $\delta_{\text{global}} \leftarrow \delta$ and $q_{\text{global}} \leftarrow q$.

   (v) If $\delta > d(qb, c)$ and $m > M$ then set $m \leftarrow 0$, $q \leftarrow q_{\text{global}}$, and do the following: (jump)

   - Choose a random $\sigma \in \Sigma_n$ and set $q \leftarrow \sigma(q)$.
   - do
     - Choose a random $\sigma \in \Sigma_n$ and set $q \leftarrow \sigma(q)$,
     - choose a random $x \in [0, 1)$,

     while $x < 1/2$.

3. **return** q.

The two most crucial factors determining the runtime of Algorithm 1 are

1. How often is $d(\sigma(q_k)b, q_k b) = 1$?
2. How often will the sequence 'jump' at Step 2(iii)?

Suppose that for randomly chosen $q, b \in \mathcal{A}_N^*$ with $n = |q| = |b|$ and randomly chosen $\sigma \in \Sigma_n$, the probability that $d(\sigma(q)b, qb) = 1$ is $p(N, n)$. And let $J(N, n)$ denote the expected number of total jumps for randomly chosen c, b, q_0 satisfying the input conditions. Assuming that all improvements [jumps] will improve [worsen] the distance by 1, the expected number of times that Step 2(i) will be executed could be approximated as $\mathcal{O}\left(\frac{M}{p(N,n)}(n + 2J(N, n))\right)$. The cost of executing Steps 2(i) through 2(iv) is dominated by the distance calculations which require computing the tableaux of $\sigma(q)b$, at a cost of $\mathcal{O}(n^2)$. We also have that $M = D(qb) = \mathcal{O}(N^3)$. This gives a total number of operations of about $\mathcal{O}\left(N^3 n^2 (n + 2J(N, n))/p(N, n)\right)$.

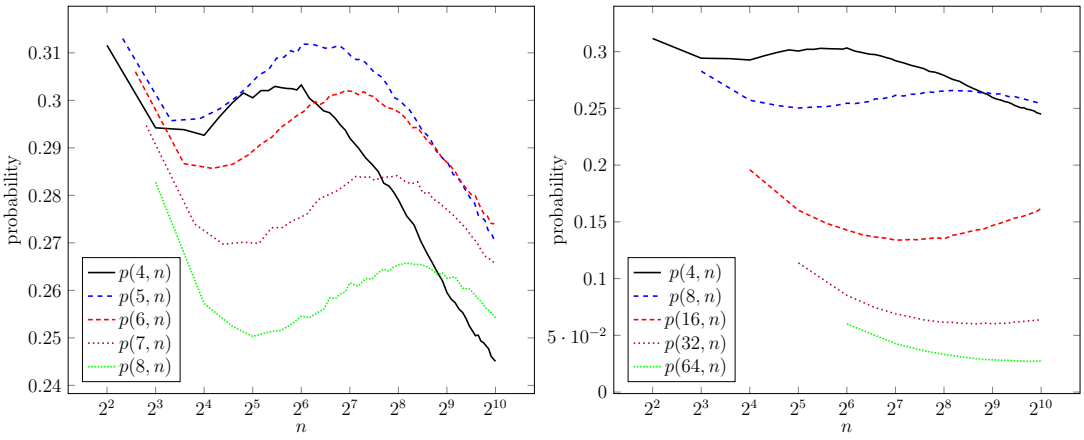

Figure 3: *Left:* functions $p(N, n)$, for $N = 4, 5, 6, 7, 8$, illustrating similar 'shapes'. *Right:* functions $p(N, n)$ for $N = 4, 8, 16, 32, 64$.

It is straightforward to experimentally approximate the function $p(N, n)$ for various values of N and n. We have done this, and summarized the results in Figure 3. It seems plausible that for each fixed value of N, the function $p(N, n)$ decays at a rate proportional to $1/(N \log n)$. It seems very likely, though, that $p(N, n) > \frac{1}{Nn}$, which is a sufficient estimate for our purposes. The function $J(N, n)$ is less straightforward to experimentally approximate by means other than running the algorithm itself and counting the number of jumps. But the experimental evidence obtained by doing so was inadequate to even formulate a conjecture.

Experimental results from this algorithm will be given in the next section, for comparison with the technique given there.

5 Lifting

In this section, we present an approach which uses Algorithm 1 combined with an algebraic observation which seems to improve the runtime for larger alphabet sizes; again, we have

only experimental evidence to suggest that it is so. The following lemma is easy to prove, and can be found as Lemma 5.4.4 in [3].

Lemma 5.1. *Let $I \subset \mathcal{A}_N$ be an interval, and for $w \in \mathcal{A}_N^*$ let $\pi_I(w)$ denote the word obtained from w be removing symbols not in I. If $w \in \mathcal{A}_N^*$ and $[w] = [w']$ then $[\pi_I(w)] = [\pi_I(w')]$.*

We will make use of this observation as follows: for each $1 \leq k \leq N$, we have that $\mathcal{A}_k$ is an interval in $\mathcal{A}_N$. To simplify notation, let $\pi_k = \pi_{\mathcal{A}_k}$. Since $[q][b] = [c]$ has a solution, it follows that $[\pi_k(q)][\pi_k(b)] = [\pi_k(c)]$ has a solution for each $1 \leq k \leq N$. We first find $q^{(1)} \in \mathcal{A}_1$ such that $[q^{(1)}][\pi_1(b)] = [\pi(c)]$, which amounts to setting $q^{(1)}$ to be the word with $|c|_1 - |b|_1$ copies of the symbol 1. Inductively, if $k < N$ and $[q^{(k)}][\pi_k(b)] = [\pi_k(c)]$, we attempt to 'lift' this to a solution over $\mathcal{A}_{k+1}$ by inserting the correct number of copies of symbol $k+1$ and applying Algorithm 1. The resulting algorithm is summarized below.

Algorithm 2 Plactic Division with lifting

Input: $c, b \in \mathcal{A}_N$ such that $[c] = [q][b]$ for some $q \in \mathcal{A}_N^*$.
Output: A value of $q \in \mathcal{A}_N$ satisfying $[c] = [q][b]$.

1. Set q to be a word with $|c|_1 - |b|_1$ copies of symbol 1, and $k \leftarrow 1$.
2. While $k < N$ do the following:
   (i) Set $k \leftarrow k + 1$.
   (ii) Pre-pend $|c|_k - |b|_k$ copies of symbol k to the word q.
   (iii) Set $q \leftarrow$ `Algorithm1`$(\pi_{k+1}(c), \pi_{k+1}(b), q_0 = q)$.

There are several considerations that seem to contribute to a better runtime in this approach. In Algorithm 1, all of the monoid operations are performed on words of 'full length'; but here, we replace many of those operations with operations on words of smaller length. And here, each 'lifting step' begins from a q_0 with $d(q_0\pi_{k+1}(b), \pi_{k+1}(c)) \leq |c|_{k+1}$, which is relatively small. Moreover, it seems to happen some good portion of the time that a solution can be obtained by simply relocating copies of the symbol $k+1$, which is relatively easy as the distance to a solution can be decreased monotonically in that case, reducing the number of 'jumps' that are needed. Every solution to $[q][b] = [c]$ is a solution to $[\pi_k(q)][\pi_k(b)] = [\pi_k(c)]$, but the latter generally has more solutions than the former; so the lifting of some solutions does require permuting several different symbols.

We implemented Algorithms 1 and 2 in C, building on the code generously made available by Daniel Brown in [1], and the source code is available on our web site. For each indicated alphabet size N, and each indicated word size $n = |a| = |b|$, we performed the following experiment 100 times: choose random words $a, b \in \mathcal{A}_N^*$ of length n, set $c = ab$, then apply Algorithm 1 to c, b. For each value of N and n, we found the median runtime required to find a solution; in all cases, the solutions were verified to be correct. The experiments described here were run on a single core of an Intel® Core™ i7 running at 3.10GHz, and the resulting wall-clock times are reported. These results are summarized in Figure 4.

The division challenge problem given in [1] was, given the following values of $d, b \in \mathcal{A}_{64}^*$ (with `base64` binary-to-text encoding), find $q \in \mathcal{A}_{64}^*$ such that $d = qb$.

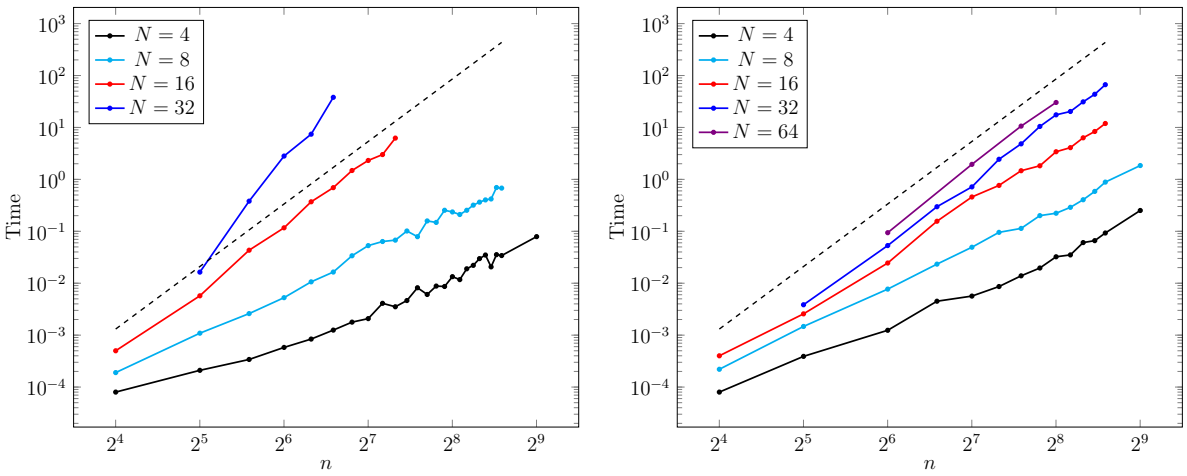

Figure 4: *Left:* median timing results for Algorithm 1, with logarithmic n and t axes. *Right:* median timing results for Algorithm 2, with logarithmic n and t axes. For reference, both plots include the same curve $t = 5 \times 10^{-7} n^4$, as a dashed line. Note that Algorithm 1 was not run with $N = 64$ because relevant values of n would begin at $n = 64$, and it was deemed impractical to obtain enough points for meaningful results.

```
d=
xvupnymwzktxyjsuxipsuxgopstekorrxzcjloqvybhkmntxaeikmqwyYbgilpvxyzVZdhkouvvxzUXbfhnqstvxzTWaeglp
rsstySVZbfhmoorsxxxzRRVZacjmmnruuwwPPTXZbejkmnttuvOOQVXabdjllmnquvwMNPTUXYchkkllltuvxLMOOSVXZbgi
jjkqqtvwxKLMNOSWYYbhiiioopuvwJKKMMRUWXabffghjltuuxIJJKLPTUWZacdefiiiksvHHIJKOQRUWXXbbccghjrtvwxG
GHHILLPSSUVVWZbegijosuwyFFGGHIKNOQTUUVXaaegiinqtxxxDEEFGGJMMOOSSTWYYbeggmnouuuxCDDDFFGJLLMNRRSTW
WZffgiknstvvwyABBCDEFHIKLMNOQRUVYbbdhijooootuz9AABBCEFFHJKLNOQTUVZaadfiklllmpqswx89999BCDDDHHKKL
MNSTTTYaacchhhjkoqtv788889BCCCFGIIJKKPRRRUUWZZbbffgjnooqqy67777888ABCFFFGHHMNNNQSVYYYabdfhiklooo
orx56666777999ABEFGGJJKLMPPQRUUYaddggiimnnoswwwy44555666788 9BBDDDGHILLMOOQRSTUVXYdddllnnqrrss33
33344455778888BBEGGGIKLNNNNNQUVVVZccfhikmmmrsww222222333446777 79AABBDDGJJJJJKLMNTTUUWdffggijknpu
w11111122222234455 99AAAAABCCCDGHIJPQSTUUWYZbcghilmmpvvw0000000111111223333555788ABBCCDEIJLMNNQQR
Rabbbccdhhijkmrrsv//////////0000001113333566666789ACGJJJKMMOOSTTVZZZaaaaglmpqrrvvwwxz++++++++++
++++++++++++/11333788BBBCDHJJJJJNNNNNOOOOOPPPTTWXYYYbcefkklpwzz

b=
gnxkOR2uN/j/sWwxNHcMKh1DaMaV4ifNUkJhjr9WWVAHVA5FNRdNYt1/bNGYuq5lZIjIiGtxjdVlT+shNNf5NnYWawpQPJZS
txH376j3JQgqwLLy0o5dq4vLeUJrSyoNUGfFB+dQawvYYRTQH+tJZQiAusuD+VTNYkqBoVnl0Vt2CDKGNhNCdiYzf7O6IhgM
JVmQxgmAGUPQwOinni6asO+sqodfogSB4BODg3UTUl5xnDOALslyiSm3A7vO+8kOr8976RCTf19I+ZGWfihspGQUdcCwrcCm
YRow31AEWMwbKnPLD41maUNHsOBVtJJU58RZcjubTZqnga1f13Bsb/lLnOrXg73vDhCEpbr+yUi6ZOYc+mZW+hB2Cvih6vJ3
km3wxaMag86a2i+k1r9dOmcKTITJwjONvr1mDlpISCsmMwTbMcE7ddvbVjGw+TpP9xqQPaOTBMRkW2zP8zcc+8kAr1XClSeb
3LKBriZYkHY8OP7zaDJh3JJNNgwY/lpf
```

In 503 seconds on a single core of an i7 @ 3.10GHz processor, the method above found the solution

```
q=
DpveA9nksHV7BoR28kKx+7GYpxV8xOOFpkx8gyKTwF9+uu7UCqbmloOsbX3jFh3/xXPjmMG1r+VumFPtObboyteGsHMikz7l
29TqOvJnZyiL8F2yzGI6ithFzhgJT+VWmPLsIhXSxjxztvhicasvxb28Mmu/BHvkviiKX1nbZJud6Wb8ai6r+M7MqK2L/8z2
NK2/8XFS9xDIUuZEU9uScHBDWaSfYllXjelkoAfKJgNaROQ6H9SODXo3URMFZUopGbD8YGgNHXN1CLLTsTiKAjxJVvxZOooz
wCyPab1oaCF7vBT4tKx8nwotrf/C9bo7nwm2u1OLBWuGY4o+jvp3U4cRuG+USHi8eibQwgxaIrBgvoL5H90V+OFTynBB7b1l
2d+ES48+3t5xiZ15+MgBI3NdM6ow7JuKQULGJtqmkqu3GE9uWgJD7vKYO4e3+Am5JRbCqkk6xUohtFAlcOB7fw/XOJbLJe8R
vKDvUNJolcTeO5ElwMkOf53BRhZPbAjf
```

5.1 Speculation

We have no reasonable explanation for why Algorithm 2 appears to be faster than Algorithm 1, but we can speculate. It appears that the number of jumps in Algorithm 1 might grow like $J(N,n) = \mathcal{O}(n^{h(N)})$, for some function h. If the expected number of jumps needed while lifting is much smaller, say $\mathcal{O}(n)$ for $|\pi_k(b)| = n$, we can approximate the number of operations needed by Algorithm 1 to perform the lift from $\mathcal{A}_k$ to $\mathcal{A}_{k+1}$. The inputs b, q over $\mathcal{A}_{k+1}$ would have expected length $n(k+1)/N$, and the initial distance $d(qb, c)$ would be no more than n/N, corresponding to the expected number n/N of copies of symbol $k+1$ just inserted. Thus, the distance $d(qb, c)$ in Algorithm 1 would need to be improved about $n/N + 2n(k+1)/N$ times. With those assumptions and the using the notations from Section 4, the number of times Step 2(i) of Algorithm 1 would be executed should be about

$$\mathcal{O}\left(\frac{M}{p(k+1, \frac{n(k+1)}{N})}\left(\frac{n}{N} + \frac{n(k+1)}{N}\right)\right) = \mathcal{O}\left(\frac{(k+1)^6 n^2}{N^2}\right).$$

Again, the number of operations is dominated by the time required to compute the tableaux, in this case $\mathcal{O}(n^2(k+1)^2/N^2)$. So the total number of operations needed to perform the lift from $\mathcal{A}_k$ to $\mathcal{A}_{k+1}$ would be about $\mathcal{O}\left((k+1)^8 n^4/N^4\right)$. Summing these from $k = 1$ to $k = N-1$ would give a total number of operations of about

$$\sum_{k=1}^{N-1} \mathcal{O}\left((k+1)^8 n^4/N^4\right) = \mathcal{O}\left(n^4 N^5\right).$$

While this estimate appears to compare favorably with the experimental evidence with respect to n, we again point out that it is purely speculation.

6 Remarks

Precisely analyzing the runtime of the algorithms here seems to be challenging. It might, perhaps, be easier to produce a deterministic variant for which an analysis is more straightforward. But every attempt we made to do so seemed to have worsened the time to something obviously exponential.

The variance of the runtimes for these algorithms seems to be quite high. Some preliminary experiments suggest that this is due more to the sequence of random choices than the particular inputs. This could potentially be leveraged, by producing a variant which automatically restarts if too high a percentage of the total time has been spent without obtaining any improvement.

7 Acknowledgements

We are very grateful to Daniel Brown for multiple email conversations and taking the time to consider the ideas here and provide feedback. Several of his suggestions led to simplifications of the overall presentation, as well as a simplification of Proposition 3.1 and its proof. We are also very pleased to contribute this paper in honor of Joachim Rosenthal's 60th birthday.

References

[1] Daniel R. L. Brown. Plactic key agreement (insecure?). Cryptology ePrint Archive, Paper 2021/625, 2021. `https://eprint.iacr.org/2021/625`.

[2] Donald E. Knuth. Permutations, matrices, and generalized Young tableaux. *Pacific Journal of Mathematics*, 34(3):709 – 727, 1970.

[3] M. Lothaire. *Algebraic Combinatorics on Words*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2002.

[4] Muhammad Rabi and Alan T. Sherman. Associative one-way functions: A new paradigm for secret-key agreement and digital signatures. Technical report, USA, 1993.

[5] C. Schensted. Longest increasing and decreasing subsequences. *Canadian Journal of Mathematics*, 13:179–191, 1961.

[6] P.W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.