Pollard's Rho for Discrete Logarithms

November 28, 2016

Pollard's Rho for Discrete Logarithm is a variant of J.M. Pollard's original Pollard's Rho algorithm used for factoring. Like the original algorithm used for factoring, the Rho method for solving the discrete logarithm takes advantage of what is commonly known as the birthday paradox. The birthday paradox states given N uniformally distributed, numbered elements, one only needs to randomly choose $\approx \sqrt{N}$ elements in the set to likely end up picking an element twice.

It is a probabilistic algorithm that is easily parallelize, though doing so only yields linear increases in speed.

0.1 The Basic Idea

Given the discrete log problem:

- Let g be a generator for the finite cyclic group G, which has an order of N.
- For some integer x, where $0 \leq x \leq N$, $g^x = h$ where h is a an element from G.
- The integer x is called the discrete logarithm of h to generator g, denoted $log_g h$.

We choose integers a, b in the range of $[0,\ N-1]$ such that:

- We obtain a random element of G, $g^a h^b$.
- We compute random group elements until we get two elements, $g^{a_i} h^{b_i}$ and $g^{a_j} h^{b_j}$, where $a_i b_i \equiv a_j b_j \pmod{N}$.
- We can then derive the equation: $x \equiv (a_j - a_i)(b_i - b_j)^{-1}$ for $b_i \neq b_j$.

0.2 The problem with the naive approach

If we simply consider the basic approach above, it's simple to see that the space complexity of the solution would scale directly with the time complexity, as we're looking for a repeat element in any one of the expected $\approx \sqrt{N}$ elements we previously generated.

The Rho method minimizes storage requirement by utilizing Floyd's Cycle Finding algorithm [1]. Floyd's algorithm uses two pointers (thus constant space) to traverse a periodic sequence where one pointer takes one step (the "tortoise" pointer) while the other pointer takes two (the "hare" pointer). Donald E. Knuth's theorem on Floyd's algorithm claims a repeat element can be found in the sequence within a minimum range of iterations (i) where i lies in the range of $\mu \leq i \leq \mu + \lambda$ [2].

0.3 The algorithm

Let:
G be a cyclic group of order p
$g, h \in G$, and a partition of $G = S_0 \cup S_1 \cup S_2$
$f : G \to G$ be a map:

1. Initialize tortoise pointer $a_0 \leftarrow 0, b_0 \leftarrow 0, x_0 \leftarrow 1 \in G$.
2. Initialize hare pointer $A_0 \leftarrow 0, B_0 \leftarrow 0, X_0 \leftarrow 1 \in G$.
3. Initialize $i \leftarrow 1$.
4. Let

$$f(x_i, a_i, b_i) = \begin{cases} x_{i+1} = hx_i, a_{i+1} = a_i, b_{i+1} = b_i + 1 \pmod p & if|x_i \in S_0 \\ x_{i+1} = x_i^2, a = 2a, b = 2b & if|x_i \in S_1 \\ x_{i+1} = gx_i, a_{i+1} = a_i + 1 \pmod p, b_{i+1} = b_i & if|x_i \in S_2 \end{cases}$$

5. Loop until $i = p$:

$$x_{i+1}, a_{i+1}, b_{i+1} = f(x_i, a_i, b_i)$$
$$X_{i+1}, A_{i+1}, B_{i+1} = f(f(X_i, A_i, B_i))$$

```
if $X = x$:

if $b - B = 0$:
return fail

else:
return $(b - B)^{-1}(A - a) \pmod p$
```

0.4 Possible improvements for Pollard's Rho

Because Pollard's Rho looks for a single collision in the same group, several processors can be used to parallelize if communication between the processors is fostered correctly. One method to do this is with the use of *distinguished points*.

Distinguished points in the context of our problem, are going to be group elements that have some sort of easily checkable property. Numbers that have a fixed number of leading zeroes is a common property used. The property is adjusted with respect to the given problem, in such a way so that there are enough distinguished points reduce the number of steps taken before cycle in the sequence is reached, but not so many points so that storage of these points would become a problem. The actual process of parallelizing Pollard's Rho is as follows:

- Each processor initialize with different exponenents $a_0, b_0 \in [0, p)$ where p is the order of the group.
- A hash table is used to store triples consisting of $\{x_i, a_i, b_i\}$, x_i being the key, where $x_i = g^{a_i} h^{b_i}$
- Each processor then iterates through algorithm normally, storing the triples in the hash table if x_i fulfills the distinguishable property requirement. If a hash collision is encountered, then the result is calculated from the other collision values $(b_i - b_j)^{-1}(a_j - a_i) \pmod p$

0.5 Analysis of Pollard's Rho

On a single processor, a collision is expected to occur in $\sqrt{\pi p/2}$ iterations [PR for DLP]. When the work is divided among m processors generating points independently, this time can be reduced linearly to $\sqrt{\pi p/2}/m$. If we denote the probability of hitting a distinguished point among our iterations as θ, the number of steps from a collision to its detection is a geometrically distributed (as we're simply looking for the first distinguished point after a collision occurs) random variable with an expected value of $1/\theta$. Thus the overall runtime expected runtime of parallelized Pollard's Rho would be:

$$\sqrt{\pi p/2}/m + 1/\theta$$