Factoring Algorithm Based on Parameterized Newton Method

Zhengjun Cao and Lihua Liu

Abstract. Given a non-square $n = p\,q$, since p, q are two roots of $x^2-\theta x+n = 0$, where $\theta = p + q$ is unknown, one can pick the initial values l, r and use Newton method to construct $A(\theta, l, k), B(\theta, r, k)$ approximating to p, q, respectively, where k is the iteration depth. Solve $A(\theta, l, k)B(\theta, r, k) = n \wedge \theta > 2\sqrt{n} \wedge l < A(\theta, l, k) < \sqrt{n} < B(\theta, r, k) < r$ to obtain the approximations of θ. Accumulate and sort the approximations for different initial values. Then pivot these approximations to search for the target θ such that $\theta^2 - 4n$ is a square. The success probability of this algorithm depends on the choice of initial values, the iteration depth, and the search scope around the pivots. The algorithm can be easily parallelized, and its complexity can be restricted to $O(\log^9 n)$.
Keywords: Factorization; Parameterized Newton method.

1 Introduction

In 1931, Lehmer and Powers [2] introduced an interesting factoring algorithm, called the continued fraction method. The quadratic sieve was introduced and developed by Pomerance *et al.* [4, 6]. The elliptic curve method [3] and number field sieve [1, 5] are quite practical. Most of these modern factoring algorithms are based on the fact: if $a^2 = b^2 \bmod n$, then $\gcd(a + b, n)$ or $\gcd(a - b, n)$ could be a nontrivial factor of n. We refer to [7] for some interesting discussions on these algorithms. In this paper, we introduce a factoring algorithm based on parameterized Newton method. Though it runs in polynomial time, its success probability seems hard to decide theoretically. Some experimental instructions will be explicitly discussed.

2 The basic method

Suppose $n = p\,q$, $p < q$, and p, q are two different positive integers. Let $\theta = p + q$. We consider the parameterized equation $x^2 - \theta x + n = 0$, where θ is unknown. Using Newton method, we obtain the iteration formula

$$x_{i+1} = x_i - \frac{x_i^2 - \theta x_i + n}{2x_i - \theta} = \frac{x_i^2 - n}{2x_i - \theta}, \quad i = 1, 2, \cdots \tag{1}$$

Z. Cao is with Department of Mathematics, Shanghai University, Shanghai, China. caozhj@shu.edu.cn
L. Liu is with Department of Mathematics, Shanghai Maritime University, Shanghai, China. liulh@shmtu.edu.cn

For the initial values l, r, and iteration depth k, one can obtain the approximations $A(\theta, l, k), B(\theta, r, k)$ of p, q, respectively. Both are rational fractions in θ. Solve the equation

$$A(\theta, l, k)B(\theta, r, k) = n \tag{2}$$

to obtain some approximations of θ. Then test each approximation $\hat{\theta}$ by checking whether $\lambda^2 - 4n$ is a square for some integer $\lambda \in [\hat{\theta} - \kappa, \hat{\theta} + \kappa]$, where κ is a preset bound.

For example, $n = 59 \times 79$, the target number is 138. After 3 iterations, the constructed fraction is quite complicated. See Figure 1 for some tests, where the initial values are 30 and 140. Clearly, the algorithm's performance depends heavily on the choice of initial values and iteration depth. So, the crude method cannot be practically implemented.

Figure 1: An example for the basic method

```
n = 4661; f[x_] := (x^2 - n) / (2 * x - a); Together[Nest[f, x, 3]]
(-471 972 192 456 241 + 607 559 140 686 a^2 - 108 624 605 a^4 + 4661 a^6 - 2 430 236 562 744 a x + 695 197 472 a^3 x - 37 288 a^5 x +
   2 835 275 989 868 x^2 - 1 824 893 364 a^2 x^2 + 130 508 a^4 x^2 + 2 433 191 152 a x^3 - 261 016 a^3 x^3 - 1 520 744 470 x^4 + 326 270 a^2 x^4 - 261 016 a x^5 + 130 508 x^6 - x^8) /
 ((a - 2 x) (-9322 + a^2 - 2 a x + 2 x^2) (43 449 842 - 18 644 a^2 + a^4 + 55 932 a x - 4 a^3 x - 55 932 x^2 + 6 a^2 x^2 - 4 a x^3 + 2 x^4))

Solve[Nest[f, 30, 3] * Nest[f, 140, 3] == n, a] // N
{{a → -123.687}, {a → -87.6044}, {a → -35.2378}, {a → 23.4979}, {a → 77.9092}, {a → 118.807}, {a → 169.089}, {a → 175.574},
 {a → 171.552 - 3.04217 i}, {a → 171.552 + 3.04217 i}, {a → 174.04 - 2.67069 i}, {a → 174.04 + 2.67069 i}, {a → 175.233 - 1.40399 i}, {a → 175.233 + 1.40399 i}}

Solve[Nest[f, 30, 4] * Nest[f, 140, 4] == n, a] // N
{{a → -133.697}, {a → -125.282}, {a → -111.658}, {a → -93.4059}, {a → -71.3026}, {a → -46.2832}, {a → -19.3969}, {a → 8.24364}, {a → 35.5181},
 {a → 61.3557}, {a → 84.7875}, {a → 104.974}, {a → 121.151}, {a → 132.31}, {a → 169.579}, {a → 175.413}, {a → 170.275 - 1.83293 i}, {a → 170.275 + 1.83293 i},
 {a → 171.655 - 2.71105 i}, {a → 171.655 + 2.71105 i}, {a → 172.945 - 2.80114 i}, {a → 172.945 + 2.80114 i}, {a → 173.933 - 2.46902 i}, {a → 173.933 + 2.46902 i},
 {a → 174.628 - 1.93781 i}, {a → 174.628 + 1.93781 i}, {a → 175.079 - 1.31933 i}, {a → 175.079 + 1.31933 i}, {a → 175.332 - 0.665617 i}, {a → 175.332 + 0.665617 i}}

Print[59 * 79, "    ", 59 + 79]
4661    138
```

3 The new algorithm

To polish the basic method, one should restrict the iteration depth. Practically, it is better to set the depth $k \leq 3$. By changing the initial values flexibly, the weakness of small depth could be compensated. Besides, one should take enough initial values to obtain enough approximations, and delete those obvious improper approximations. To do so, we use the below restrictions

$$A(\theta, l, k)B(\theta, r, k) = n \ \wedge \ \theta > 2\sqrt{n} \ \wedge \ l < A(\theta, l, k) < \sqrt{n} < B(\theta, r, k) < r \tag{3}$$

The new algorithm can be illustrated below (Fig. 2, Wolfram Mathematica code), where $n = 336774871 = 14591 \times 23081$, the target number is 37672, and the nearest approximation is 37681.

Figure 2: An example for the new method

```
MyFactor[mynumber_, myleft_, myright_, IterationDepth_, StepLength_] :=
 (n = mynumber; b = IntegerPart[Sqrt[n]]; B = {};
  f[x_] := (x^2 - n) / (2 * x - a); g[x_, k_] := Nest[f, x, k];
  l = myleft; r = myright; K = IterationDepth; H = StepLength;
  For[i = 1, i < Length[K] + 1, i++, k = K[[i]];
   For[j = 1, j < Length[H] + 1, j++, d = H[[j]];
    For[u = 0, u < IntegerPart[(r - l) / d], u++, s = l + u * d;
       If[s > b, Break[],
          For[v = 0, v < IntegerPart[(r - l) / d], v++, t = r - v * d;
             If[t < b, Break[], left = g[s, k]; right = g[t, k];
                T = N[Solve[left * right == n, a], 10]; T = Select[a /. T, Im[#] == 0 && # > 2 * Sqrt[n] &];
                T = Select[T, s < (left /. a → #) < b < (right /. a → #) < t &];
                B = Union[B, T]; Clear[T]]]]]]];
  B = Function[x, IntegerPart[x]] /@ B;
  B = Union[B]; Print[B]; Print[Differences[B]])
```

```
MyFactor[336774871, 10000, 30000, {1}, {450, 700}]
```

```
{36734, 36748, 36751, 36780, 36823, 36844, 36914, 36954, 36963, 37026, 37055, 37062, 37145, 37156, 37170, 37183, 37295,
 37300, 37495, 37498, 37543, 37563, 37624, 37721, 37769, 37798, 37887, 37911, 37921, 37996, 38079, 38215, 38243, 38257, 38280, 38302,
 38475, 38570, 38639, 38671, 38732, 38882, 38932, 39018, 39041, 39181, 39368, 39415, 39421, 39650, 39781, 39840, 40096, 40367, 40549, 40824}
{14, 3, 29, 43, 21, 70, 40, 9, 63, 29, 7, 83, 11, 14, 13, 112, 5, 195, 3, 45, 20, 61, 97, 48, 29, 89,
 24, 10, 75, 83, 136, 28, 14, 23, 22, 173, 95, 69, 32, 61, 150, 50, 86, 23, 140, 187, 47, 6, 229, 131, 59, 256, 271, 182, 275}
```

```
MyFactor[336774871, 10000, 30000, {1, 2}, {450, 700}]
```

```
{36732, 36734, 36748, 36751, 36755, 36756, 36778, 36780, 36815, 36817, 36823, 36844, 36848, 36851, 36857, 36894, 36914, 36949, 36951, 36954, 36963, 37015, 37026,
 37037, 37055, 37062, 37070, 37076, 37116, 37118, 37145, 37156, 37170, 37183, 37188, 37196, 37207, 37252, 37271, 37295, 37300, 37308, 37352, 37412, 37454, 37490,
 37495, 37498, 37509, 37543, 37563, 37599, 37619, 37620, 37624, 37681, 37721, 37741, 37743, 37766, 37769, 37798, 37858, 37887, 37889, 37895, 37909, 37911, 37921,
 37966, 37996, 38030, 38066, 38079, 38118, 38160, 38164, 38215, 38243, 38257, 38274, 38280, 38290, 38302, 38309, 38408, 38426, 38475, 38482, 38511, 38520, 38570,
 38623, 38639, 38671, 38707, 38732, 38746, 38807, 38836, 38882, 38932, 38958, 38962, 38963, 39018, 39041, 39073, 39109, 39162, 39181, 39250, 39339, 39368, 39382,
 39415, 39421, 39498, 39507, 39582, 39648, 39650, 39781, 39792, 39807, 39840, 39928, 40033, 40096, 40187, 40334, 40367, 40418, 40549, 40650, 40725, 40824, 40875}
{2, 14, 3, 4, 1, 22, 2, 35, 2, 6, 21, 4, 3, 6, 37, 20, 35, 2, 3, 9, 52, 11, 11, 18, 7, 8, 6, 40, 2, 27, 11, 14, 13, 5, 8, 11, 45, 19, 24, 5, 8, 44, 60, 42, 36, 5, 3, 11, 34,
 20, 36, 20, 1, 4, 57, 40, 20, 2, 23, 3, 29, 60, 29, 2, 6, 14, 2, 10, 45, 30, 34, 36, 13, 39, 42, 4, 51, 28, 14, 17, 6, 10, 12, 7, 99, 18, 49, 7, 29, 9, 50, 53, 16, 32, 36,
 25, 14, 61, 29, 46, 50, 26, 4, 1, 55, 23, 32, 36, 53, 19, 69, 89, 29, 14, 33, 6, 77, 9, 75, 66, 2, 131, 11, 15, 33, 88, 105, 63, 91, 147, 33, 51, 131, 101, 75, 99, 51}
```

```
MyFactor[336774871, 10000, 30000, {1, 2, 3}, {450, 700}]
```

```
{36722, 36732, 36734, 36748, 36751, 36755, 36756, 36777, 36778, 36780, 36785, 36799, 36803, 36815, 36817, 36819, 36823, 36835, 36844, 36848, 36849, 36851, 36857,
 36863, 36868, 36894, 36914, 36919, 36943, 36949, 36951, 36954, 36963, 36971, 37010, 37015, 37021, 37026, 37037, 37055, 37062, 37067, 37069, 37070, 37076, 37110,
 37116, 37118, 37121, 37145, 37150, 37156, 37165, 37170, 37183, 37187, 37188, 37196, 37207, 37221, 37252, 37271, 37279, 37290, 37295, 37300, 37308, 37350, 37352,
 37408, 37412, 37452, 37454, 37462, 37490, 37495, 37498, 37509, 37513, 37543, 37546, 37561, 37563, 37599, 37605, 37619, 37620, 37624, 37631, 37647, 37681, 37685,
 37709, 37720, 37721, 37741, 37743, 37766, 37769, 37776, 37780, 37798, 37838, 37843, 37858, 37887, 37889, 37895, 37897, 37898, 37909, 37911, 37921, 37952, 37966,
 37996, 38005, 38030, 38054, 38066, 38079, 38100, 38118, 38143, 38160, 38164, 38182, 38208, 38215, 38243, 38257, 38262, 38274, 38280, 38290, 38302, 38305, 38309,
 38325, 38386, 38394, 38408, 38426, 38442, 38475, 38482, 38496, 38511, 38520, 38547, 38549, 38570, 38594, 38623, 38638, 38639, 38671, 38707, 38732, 38746, 38748,
 38807, 38813, 38836, 38874, 38882, 38932, 38958, 38962, 38963, 38964, 38987, 39018, 39039, 39041, 39064, 39073, 39088, 39109, 39156, 39162, 39181, 39234, 39241,
 39250, 39299, 39339, 39362, 39368, 39382, 39415, 39421, 39422, 39478, 39498, 39507, 39532, 39582, 39648, 39650, 39719, 39781, 39786, 39792, 39807, 39822, 39840,
 39850, 39911, 39918, 39928, 39969, 40033, 40096, 40187, 40273, 40334, 40338, 40367, 40400, 40418, 40549, 40580, 40650, 40679, 40725, 40824, 40825, 40875, 40889}
{10, 2, 14, 3, 4, 1, 21, 1, 2, 5, 14, 4, 12, 2, 2, 4, 12, 9, 4, 1, 2, 6, 6, 5, 26, 20, 5, 24, 6, 2, 3, 9, 8, 39, 5, 6, 5, 11, 18, 7, 5, 2, 1, 6, 34, 6, 2, 3,
 24, 5, 6, 9, 5, 13, 4, 1, 8, 11, 14, 31, 19, 8, 11, 5, 5, 8, 42, 2, 56, 4, 40, 2, 8, 28, 5, 3, 11, 4, 30, 3, 15, 2, 36, 6, 14, 1, 4, 7, 16, 34, 4, 24, 11, 1, 20, 2,
 23, 3, 7, 4, 18, 40, 5, 15, 29, 2, 6, 2, 1, 11, 2, 10, 31, 14, 30, 9, 25, 24, 12, 13, 21, 18, 25, 17, 4, 18, 26, 7, 28, 14, 5, 12, 6, 10, 12, 3, 4, 16, 61, 8, 14,
 18, 16, 33, 7, 14, 15, 9, 27, 2, 21, 24, 29, 15, 1, 32, 36, 25, 14, 2, 59, 6, 23, 38, 8, 50, 26, 4, 1, 1, 23, 31, 21, 2, 23, 9, 15, 21, 47, 6, 19, 53, 7, 9, 49, 40,
 23, 6, 14, 33, 6, 1, 56, 20, 9, 25, 50, 66, 2, 69, 62, 5, 6, 15, 15, 18, 10, 61, 7, 10, 41, 64, 63, 91, 86, 61, 4, 29, 33, 18, 131, 31, 70, 29, 46, 99, 1, 50, 14}
```

```
p = 14591; q = 23081; n = 14591 * 23081; w = p + q; Print[n, "      ", w]
```

```
336774871      37672
```

4 Performance analysis

We now give more details about experimental instructions.

On the iteration depth. We have observed that it became inefficient to solve the equation $A(\theta, l, k)B(\theta, r, k) = n$ even if the iteration depth $k = 3$. So, it is better to restrict k to $\{1, 2\}$. The loss can be compensated by assigning initial values through the sets

$$\{l, l + d, l + 2d, \cdots, l + [\frac{r - l}{d}]d\}, \quad \{r, r - d, r - 2d, \cdots, r - [\frac{r - l}{d}]d\}.$$

As we see, the nearest approximation is still 37681 for $k = 1, 2, 3$. While it is 37624 for the extreme case $k = 1$.

On the choice of step length. It is somewhat difficult to decide which step length is better than others (see Fig. 3). Practically, we suggest some random step lengths. Since the approximations are finally accumulated and sorted, the involved randomness could generate a good distribution and good differences.

Figure 3: An example for different step lengths

```
MyFactor[336774871, 10000, 30000, {1, 2}, {312, 1000}]
{36705, 36710, 36713, 36723, 36731, 36747, 36751, 36755, 36760, 36762, 36771, 36793, 36794, 36811, 36824, 36834, 36848, 36851, 36863, 36868, 36872, 36874, 36877, 36914, 36918,
 36928, 36947, 36957, 36969, 36976, 36982, 36985, 36991, 37001, 37020, 37031, 37032, 37039, 37063, 37077, 37093, 37112, 37120, 37121, 37128, 37132, 37159, 37177, 37198,
 37213, 37223, 37230, 37258, 37260, 37266, 37276, 37279, 37294, 37312, 37318, 37320, 37356, 37358, 37372, 37373, 37410, 37414, 37425, 37432, 37447, 37448, 37472, 37474,
 37492, 37504, 37514, 37517, 37520, 37529, 37551, 37585, 37591, 37594, 37648, 37655, 37663, 37669, 37674, 37693, 37708, 37754, 37761, 37765, 37787, 37789, 37830, 37843,
 37852, 37862, 37881, 37902, 37957, 37966, 37969, 37971, 37972, 37977, 37986, 37996, 38036, 38037, 38038, 38039, 38054, 38067, 38098, 38135, 38155, 38162, 38164, 38190,
 38213, 38231, 38257, 38264, 38279, 38287, 38303, 38330, 38337, 38347, 38357, 38392, 38424, 38433, 38439, 38470, 38481, 38507, 38516, 38538, 38546, 38558, 38595, 38601,
 38633, 38654, 38671, 38708, 38725, 38743, 38768, 38770, 38803, 38812, 38817, 38818, 38897, 38912, 38968, 38970, 38978, 39009, 39031, 39037, 39055, 39078, 39079, 39102,
 39147, 39185, 39191, 39271, 39277, 39286, 39327, 39331, 39339, 39360, 39385, 39450, 39466, 39479, 39557, 39563, 39570, 39597, 39640, 39658, 39660, 39681, 39760, 39774,
 39820, 39845, 39856, 39929, 39949, 39957, 39964, 39986, 40034, 40072, 40135, 40233, 40277, 40301, 40407, 40462, 40510, 40580, 40609, 40630, 40780, 40781, 40879, 40885}
{5, 3, 10, 8, 16, 4, 4, 5, 2, 9, 22, 1, 17, 13, 10, 14, 3, 12, 5, 4, 2, 3, 37, 4, 10, 19, 10, 12, 7, 6, 3, 6, 10, 19, 11, 1, 7, 24, 14, 16, 19, 8, 1, 7, 4, 27, 18, 21, 15, 10, 7, 28, 2, 6, 10, 3, 15,
 18, 6, 2, 36, 2, 14, 1, 37, 4, 11, 7, 15, 1, 24, 2, 18, 12, 10, 3, 3, 9, 22, 34, 6, 3, 54, 7, 8, 6, 5, 19, 15, 46, 7, 4, 22, 2, 41, 13, 9, 10, 19, 21, 55, 9, 3, 2, 1, 5, 9, 10, 40, 1, 1, 1, 15,
 13, 31, 37, 20, 7, 2, 26, 23, 18, 26, 7, 15, 8, 16, 27, 7, 10, 10, 35, 32, 9, 6, 31, 11, 26, 9, 22, 8, 12, 37, 6, 32, 21, 17, 37, 17, 18, 25, 2, 33, 9, 5, 1, 79, 15, 56, 2, 8, 31, 22, 6, 18,
 23, 1, 23, 45, 38, 6, 80, 6, 9, 41, 4, 8, 21, 25, 65, 16, 13, 78, 6, 7, 27, 43, 18, 2, 21, 79, 14, 46, 25, 11, 73, 20, 8, 7, 22, 48, 38, 63, 98, 44, 24, 106, 55, 48, 70, 29, 21, 150, 1, 98, 6}
MyFactor[336774871, 10000, 30000, {1, 2}, {887, 2505}]
{36845, 36897, 37003, 37054, 37076, 37227, 37238, 37506, 37512, 37517, 37768, 37989,
 37996, 38093, 38182, 38289, 38301, 38582, 38845, 38919, 39011, 39079, 39339, 39644, 39757, 39920, 40333, 40705}
{52, 106, 51, 22, 151, 11, 268, 6, 5, 251, 221, 7, 97, 89, 107, 12, 281, 263, 74, 92, 68, 260, 305, 113, 163, 413, 372}
```

On the differences of sorted approximations. In theory, as the iteration depth k increases, the maximum of all differences will dramatically decrease. Plenty of experiments show that the maximum is generally bound above by $O(\log^5 n)$ even for the iteration depth $k = 1, 2$.

On the searching scope around each pivot. The final approximations will be used as pivots to search for the target number θ, by testing whether $\theta^2 - 4n$ is a square. Usually, one has two choices. (1) Use the mean of all approximations as the unique pivot, and choose a proper radius. (2) Select the minimum of all differences, and use the corresponding approximation as

the first pivot. If it fails, select the minimum of the remainders and update the pivot. The greedy strategy for selecting pivots is sometimes quite efficient although its programming code is somewhat complicate.

On the repeating number. In the above demonstration, we simply set the repeating number as $[\frac{r-l}{d}]$. Actually, lots of starting numbers for $l+ud, r-vd$ are wasted because they cannot generate any approximation satisfying the subsequent checking. See the below example (Fig 4), in which the repeating number is simply set as 10. While it also generates the nearest approximation 37681. In the next example, the repeating number is also set as 10. The nearest approximation is 6011287, with the difference 4615 against the target integer 6015902.

Figure 4: An example for small repeating number

```
MyFactor[mynumber_, myleft_, myright_, IterationDepth_, StepLength_] :=
 (n = mynumber; b = IntegerPart[Sqrt[n]]; B = {};
  f[x_] := (x^2 - n) / (2 * x - a); g[x_, k_] := Nest[f, x, k];
  l = myleft; r = myright; K = IterationDepth; H = StepLength;
  For[i = 1, i < Length[K] + 1, i++, k = K[[i]];
   For[j = 1, j < Length[H] + 1, j++, d = H[[j]];
    For[u = 0, u < 10, u++, s = l + u * d;
     If[s > b, Break[],
      For[v = 0, v < 10, v++, t = r - v * d;
       If[t < b, Break[], left = g[s, k]; right = g[t, k];
        T = N[Solve[left * right == n, a], 10]; T = Select[a /. T, Im[#] == 0 && # > 2 * Sqrt[n] &];
        T = Select[T, s < (left /. a → #) < b < (right /. a → #) < t &];
        B = Union[B, T]; Clear[T]]]]]]];
  B = Function[x, IntegerPart[x]] /@ B;
  B = Union[B]; Print[B]; Print[Differences[B]])

MyFactor[336774871, 10000, 30000, {1, 2}, {450, 700}]

{36755, 36780, 36815, 36823, 36851, 36894, 36914, 36949, 37055, 37062, 37070, 37076, 37145, 37170, 37183, 37196, 37207, 37271, 37300, 37352,
 37454, 37490, 37498, 37563, 37619, 37620, 37624, 37681, 37721, 37741, 37743, 37766, 37769, 37858, 37887, 37889, 37895, 37909, 37911, 37921, 37996,
 38030, 38118, 38164, 38215, 38243, 38257, 38274, 38280, 38290, 38302, 38309, 38408, 38426, 38475, 38482, 38511, 38520, 38570, 38639, 38671, 38707,
 38732, 38746, 38807, 38836, 38882, 38932, 38958, 38962, 38963, 39018, 39041, 39073, 39109, 39162, 39181, 39250, 39339, 39368, 39382, 39415, 39421,
 39498, 39507, 39582, 39648, 39650, 39781, 39792, 39807, 39840, 39928, 40033, 40096, 40187, 40334, 40367, 40418, 40549, 40650, 40725, 40824, 40875}

{25, 35, 8, 28, 43, 20, 35, 106, 7, 8, 6, 69, 25, 13, 13, 11, 64, 29, 52, 102, 36, 8, 65, 56, 1, 4, 57, 40, 20, 2, 23, 3, 89, 29, 2, 6, 14, 2, 10, 75, 34, 88, 46, 51, 28, 14, 17, 6, 10, 12, 7, 99, 18,
 49, 7, 29, 9, 50, 69, 32, 36, 25, 14, 61, 29, 46, 50, 26, 4, 1, 55, 23, 32, 36, 53, 19, 69, 89, 29, 14, 33, 6, 77, 9, 75, 66, 2, 131, 11, 15, 33, 88, 105, 63, 91, 147, 33, 51, 131, 101, 75, 99, 51}

MyFactor[8257092578401, 1000000, 5000000, {1, 2}, {10500, 35021}]

{5747823, 5748676, 5754547, 5754967, 5756967, 5759768, 5764353, 5769199, 5771712, 5779729, 5783553, 5788552, 5794804, 5797608,
 5804351, 5809580, 5819850, 5824057, 5835048, 5838236, 5844337, 5849947, 5859958, 5864548, 5875279, 5878851, 5890301, 5900055, 5905025,
 5915498, 5919451, 5930643, 5945489, 5955704, 5960039, 5970971, 5985941, 6000613, 6011287, 6026379, 6041173, 6066804, 6081721, 6122255}

{853, 5871, 420, 2000, 2801, 4585, 4846, 2513, 8017, 3824, 4999, 6252, 2804, 6743, 5229, 10270, 4207, 10991, 3188, 6101, 5610, 10011,
 4590, 10731, 3572, 11450, 9754, 4970, 10473, 3953, 11192, 14846, 10215, 4335, 10932, 14970, 14672, 10674, 15092, 14794, 25631, 14917, 40534}

p = 2118751; q = 3897151; n = p * q; w = p + q; Print[n, "      ", w]

8257092578401      6015902
```

On the computational complexity. As discussed above, for the iteration depth $k = 2$ and each pair of starting values, it needs to find the roots of a 6-degree polynomial with the coefficients in $\mathbb{Z}_{n^3}$, which takes $O(\log^2 n)$ cost. To check whether an integer is a square, it also needs $O(\log^2 n)$ cost. If we take α different steps and restrict the repeating number to β, it needs to do $\alpha^2\beta^2$ loops and obtain at most $8\alpha^2\beta^2$ approximations. For each pivot, it needs to test at most $O(\log^5 n)$ integers for squareness. Thus, the total complexity is $8\alpha^2\beta^2 \times \log^2 n \times \log^2 n \times \log^5 n \in O(\log^9 n)$. Note that the squareness testing around all pivots can be done in parallel. Likewise, the accumulation of approximations can also be done in parallel.

5 Conclusion

We present a factoring algorithm which is based on a new principle, not the usual equal-residue-pair searching. It only involves some basic numerical computations and squareness checking. Though its running time can be totally controlled, we admit its success probability is quite hard to estimate due to the involved randomness. We want to stress that the algorithm is unsuited to extremely unbalanced composite numbers (the difference $a - b$ between the biggest prime factor a and the smallest prime factor b almost equals a). For conveniences, we directly use the built-in function `Solve[]` which generates many trivial solutions (negative numbers and complex numbers), and wastes much time. But this weakness can be reasonably removed.

References

[1] J. Buhler, H. Lenstra, and C. Pomerance. Factoring integers with the number field sieve. *The Development of the Number Field Sieve*, page 5094, 1993.

[2] D. Lehmer and R. Powers. On factoring large numbers. *Bulletin of the AMS*, (37):770776, 1931.

[3] H. Lenstra. Factoring integers with elliptic curves. *Annals of Mathematics*, (126):649673, 1987.

[4] H. Lenstra and C. Pomerance. A rigorous time bound for factoring integers. *Journal of the AMS*, (4):483516, 1992.

[5] J. Pollard. Factoring with cubic integers. *The Development of the Number Field Sieve*, page 410, 1993.

[6] C. Pomerance. Analysis and comparison of some integer factoring algorithms. *Computational Methods in Number Theory, Part I*, page 89139, 1982.

[7] V. Shoup. *A Computational Introduction to Number Theory and Algebra, 2nd ed.* Cambridge Univ. Press, 2008.

Appendix The Wolfram Mathematica code for the factoring algorithm

```
MyFactor[mynumber_, myleft_, myright_, IterationDepth_, StepLength_]
:=(n=mynumber; b=IntegerPart[Sqrt[n]]; B={};
  f[x_]:=(x^2-n)/(2*x-a); g[x_, k_]:=Nest[f, x, k];
  l=myleft; r=myright; K=IterationDepth; H=StepLength;
  For[i=1, i<Length[K]+1, i++, k=K[[i]];
    For[j=1, j<Length[H]+1, j++, d=H[[j]];
      For[u=0, u<IntegerPart[(r-l)/d], u++, s=l+u*d;
            If[s>b, Break[],
        For[v=0, v<IntegerPart[(r-l)/d], v++, t=r-v*d;
              If[t<b, Break[],left=g[s, k]; right=g[t, k];
          T=N[Solve[left*right==n, a], 10];
          T=Select[a/.T, Im[#]==0 && #>2*Sqrt[n] &];
          T=Select[T, s<(left/.a->#)<b<(right/.a->#)<t &];
          B=Union[B, T]; Clear[T]]]]]]];
  B=Function[x, IntegerPart[x]]/@B; B=Union[B];
  Print[B]; Print[Differences[B]])

MyFactor[8257092578401, 1000000, 5000000, {1, 2}, {10500, 35021}]
```